

AD-A284 307



①

# NAVAL POSTGRADUATE SCHOOL Monterey, California



## THESIS

DTIC  
ELECTE  
SEP 09 1994  
S B D

**Data Link Level Interconnection  
of Remote Fiber Distributed Data  
Interface Local Area Networks  
(FDDI LANs) Through the  
Critical Data Link (CDL)**

by

**Selcuk Karayakaylar**

June 1994

Thesis Advisor:

**Shridhar B. Shukla**

Approved for public release, distribution is unlimited.

**94-29487**



DTIC QUALITY INSPECTED 3

0 4 7

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

June 1994

3. REPORT TYPE AND DATES COVERED

Master's Thesis

4. TITLE AND SUBTITLE

DATA LINK LEVEL INTERCONNECTION OF REMOTE FIBER  
DISTRIBUTED DATA INTERFACE LOCAL AREA NETWORKS  
(FDDI LANs) THROUGH THE CRITICAL DATA LINK (CDL)

5. FUNDING NUMBERS

6. AUTHOR(S)

Karayakaylar, Selcuk

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School  
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION  
REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING  
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the  
official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis deals with the features and performance of a network interface device to interconnect two remote Fiber Distributed Data Interface (FDDI) Local Area Networks (LANs) through the Critical Data Link (CDL) which is a full-duplex, jam-resistant, point-to-point microwave communications system for use in imagery and signals intelligence collection systems. In particular, OPNET, a commercially available network engineering tool is used to model a medium access level remote bridge interface connecting two LANs. The effectiveness of two different load balancing techniques used to distribute traffic over the multiple channels of the CDL has been studied. Also, the effect of different jamming patterns on the bit error rate seen by the users has been studied.

DTIC QUALITY INSPECTED 3

14. SUBJECT TERMS

FDDI, LAN, CDL, MAC, LLC, bridge, simulation

15. NUMBER OF PAGES

187

16. PRICE CODE

17. SECURITY CLASSIFICATION  
OF REPORT

UNCLASSIFIED

18. SECURITY CLASSIFICATION  
OF THIS PAGE

UNCLASSIFIED

19. SECURITY CLASSIFICATION  
OF ABSTRACT

UNCLASSIFIED

20. LIMITATION OF ABSTRACT

UL

Approved for public release; distribution is unlimited

**Data Link Level Interconnection of Remote Fiber  
Distributed Data Interface Local Area Networks  
(FDDI LANs) Through the Critical Data Link (CDL)**

by

Selcuk Karayakaylar  
Lieutenant Junior Grade, Turkish Navy  
B.S.E.E., Turkish Naval Academy, 1988

Submitted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**

June 1994

Author:

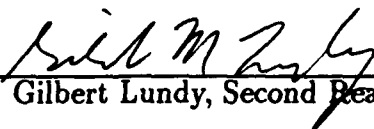


Selcuk Karayakaylar

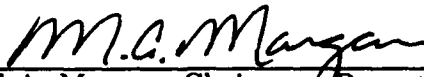
Approved by:



Shridhar B. Shukla, Thesis Advisor



Gilbert Lundy, Second Reader



Michael A. Morgan, Chairman, Department of Electrical  
and Computer Engineering

## ABSTRACT

This thesis deals with the features and performance of a network interface device to interconnect two remote Fiber Distributed Data Interface (FDDI) Local Area Networks (LANs) through the Critical Data Link (CDL) which is a full-duplex, jam-resistant, point-to-point microwave communications system for use in imagery and signals intelligence collection systems. In particular, OPNET, a commercially available network engineering tool is used to model a medium access level remote bridge interface connecting two LANs. The effectiveness of two different load balancing techniques used to distribute traffic over the multiple channels of the CDL has been studied. Also, the effect of different jamming patterns on the bit error rate seen by the users has been studied.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

## TABLE OF CONTENTS

I.	INTRODUCTION . . . . .	1
A.	PROBLEM STATEMENT . . . . .	1
B.	SCOPE . . . . .	1
C.	BENEFITS . . . . .	2
D.	ORGANIZATION . . . . .	2
II.	INTERCONNECTION OF FDDI LOCAL AREA NETWORKS THROUGH CDL . . . . .	3
A.	OVERVIEW . . . . .	3
B.	ISSUES IN HIGH SPEED LAN INTERCONNECTION . . . . .	3
1.	Preliminary . . . . .	3
2.	Bridges versus Routers . . . . .	5
C.	PROPOSED BRIDGING METHOD . . . . .	6
1.	Current Standards . . . . .	6
2.	Remote Bridging . . . . .	6
D.	UNIQUENESS OF THE CDL ENVIRONMENT . . . . .	8
1.	Description . . . . .	8
2.	System Requirements . . . . .	9
3.	Asymmetry between Command and Return Link . . . . .	9
4.	Considerations on Link Utilization . . . . .	10
5.	The Effects of Jamming . . . . .	11
III.	MODELING CDL NETWORK INTERFACE IN OPNET . . . . .	13
A.	OVERVIEW . . . . .	13
B.	REVISED FDDI LAN MODEL . . . . .	14
1.	FDDI LAN Model in Brief . . . . .	14
2.	Modifications on FDDI Station Model . . . . .	18
a.	Preliminary . . . . .	18
b.	Source Node Modifications . . . . .	18
c.	Sink Node Modifications . . . . .	19
C.	BRIDGE MODEL . . . . .	19
1.	Preliminary . . . . .	19

2.	The CPNI . . . . .	19
a.	Station Model . . . . .	19
b.	Source Node Modifications . . . . .	22
c.	MAC Node Modifications . . . . .	22
d.	Sink Node Modifications . . . . .	23
3.	The SPNI . . . . .	24
a.	Station Model . . . . .	24
b.	Source Node Modifications . . . . .	26
c.	MAC Node Modifications . . . . .	26
d.	Sink Node Modifications . . . . .	26
D.	CDL MODEL . . . . .	27
1.	OPNET Model for Point-to-Point Links . . . . .	27
a.	Preliminary . . . . .	27
b.	Transmitters . . . . .	28
c.	Receivers . . . . .	28
d.	Transceiver Pipeline Stages . . . . .	28
2.	Error Modeling over Multiple Links . . . . .	30
a.	Modifications on Error Allocation Pipeline Stage . . . . .	30
b.	Jammer Implementations . . . . .	31
E.	LAN INTERCONNECTION . . . . .	33
1.	Addressing through The Remote Bridge . . . . .	36
2.	Load Balancing over Multiple Links . . . . .	37
a.	Circular Allocation Algorithm . . . . .	43
b.	Empty Selection Algorithm . . . . .	43
F.	FAITHFULNESS OF THE MODEL . . . . .	44
IV.	MODEL TESTING . . . . .	47
A.	OVERVIEW . . . . .	47
B.	PERFORMANCE METRICS . . . . .	47
C.	TRAFFIC MONITORING . . . . .	48
1.	Overview . . . . .	48
2.	Setup . . . . .	48
3.	Results . . . . .	49
D.	RETURN LINK PERFORMANCE . . . . .	52
1.	Overview . . . . .	52

2. First Test . . . . .	52
a. Setup . . . . .	52
b. Results . . . . .	52
3. Second Test . . . . .	56
a. Setup . . . . .	56
b. Results . . . . .	56
E. COMMAND LINK PERFORMANCE . . . . .	60
V. CONCLUSIONS AND RECOMMENDATIONS . . . . .	63
A. CONCLUSIONS . . . . .	63
B. RECOMMENDATIONS . . . . .	63
APPENDIX A: CPNI SOURCE "C" CODE . . . . .	65
APPENDIX B: CPNI MAC "C" CODE . . . . .	75
APPENDIX C: CPNI SINK "C" CODE . . . . .	115
APPENDIX D: SPNI SOURCE "C" CODE . . . . .	133
APPENDIX E: SPNI MAC "C" CODE EXCERPT . . . . .	143
APPENDIX F: SPNI SINK "C" CODE . . . . .	145
APPENDIX G: CDL MODEL ERROR ALLOCATION CODE . . . . .	161
APPENDIX H: SAMPLE ENVIRONMENT FILE FOR PULSED JAMMER . . . . .	165
APPENDIX I: SAMPLE ENVIRONMENT FILE EXCERPT FOR CHANNEL-SWEPT JAMMER . . . . .	171
REFERENCES . . . . .	173
INITIAL DISTRIBUTION LIST . . . . .	175

## LIST OF FIGURES

1	Protocols relayed by the bridges and routers . . . . .	4
2	Remote MAC bridge architecture [5] . . . . .	7
3	FDDI LAN ring representation . . . . .	15
4	Ten-station FDDI LAN, <i>fddi_net_10</i> . . . . .	15
5	FDDI station model, <i>fddi_station</i> . . . . .	16
6	Source process model, " <i>fddi_gen</i> " . . . . .	16
7	MAC process model, " <i>fddi_mac</i> " . . . . .	17
8	Sink process model, " <i>fddi_sink</i> " . . . . .	17
9	Collection Platform LAN Network Interface (CPNI) . . . . .	20
10	Surface Platform LAN Network Interface (SPNI) . . . . .	25
11	The attributes used for jamming patterns . . . . .	32
12	Pulsed jammer representation . . . . .	34
13	Channel-swept jammer representation . . . . .	34
14	Interconnected network model, <i>fddi_cdl</i> . . . . .	35
15	Modified packet format, " <i>fddi_llc_fr</i> " . . . . .	35
16	Modified packet format, " <i>fddi_mac_fr</i> " . . . . .	39
17	Modified ICI packet format, " <i>fddi_mac_req</i> " . . . . .	40
18	Return link data rates and new attributes for <i>llc_sink</i> . . . . .	41
19	Command link data rate . . . . .	42
20	Local throughput of surface LAN . . . . .	50
21	Traffic directed to the command link . . . . .	50
22	Local throughput of collection platform LAN . . . . .	51
23	Throughput directed to the return link . . . . .	51
24	Accumulation of packets on the CPNI buffers with circular allocation . . . . .	53
25	Buffer overflows in the CPNI with circular allocation . . . . .	54



26	Accumulation of packets in the CPNI buffers with empty selection . .	54
27	Queing delay of the CPNI buffers with circular allocation . . . . .	55
28	Queing delay of the CPNI buffers with empty selection . . . . .	55
29	Throughput at the transmit-end of the return link . . . . .	57
30	Average BER of the return link caused by pulsed jammer . . . . .	58
31	Average BER of the return link caused by channel-swept jammer . .	58
32	Effect of pulsed jammer on the return link throughput . . . . .	59
33	Effect of channel-swept jammer on the return link throughput . . . .	59
34	Accumulation of packets in the SPNI buffer . . . . .	61
35	Queing delay of the SPNI . . . . .	61
36	Throughput of the transmit and receive-ends of the command link . .	62

## **ACKNOWLEDGMENT**

I wish to thank my thesis advisor Professor Shridhar B. Shukla for his guidance and encouragement in this research.

OPNET is a registered trademark of MIL 3 Inc.

# **I. INTRODUCTION**

## **A. PROBLEM STATEMENT**

This thesis deals with data link level interconnection of remote Local Area Networks (LANs) through Critical Data Link (CDL). The Critical Data Link (CDL) is a full-duplex, jam-resistant, point-to-point microwave communications system for use in imagery and signals intelligence collection systems. The CDL system is designed to provide a communications protocol between two or more Fiber Distributed Data Interface Local Area Networks (FDDI LAN). While the collection platform LAN is in the form of an airborne LAN that provides sensor information, the command information is provided by a ground-based LAN.

This study is concerned with the modeling of a CDL Network Interface (NI) using a commercially available network simulation program, MIL 3 Inc.'s Optimized Network Engineering Tool (OPNET).

## **B. SCOPE**

The scope of this thesis is as follows:

- Model the interconnection of FDDI LANs in the unique environment of CDL deployment.
- Evaluate the efficiency of the LAN-CDL interface.
- Establish the necessary basis for running multilink point-to-point protocol in CDL deployment.

### **C. BENEFITS**

This thesis continues the development of CDL related simulation/modeling programs started in [1]. In that study, the capabilities of OPNET's original FDDI LAN model were enhanced for the use in CDL deployment. We have added:

- A model of NI that performs load balancing over multiple links.
- A CDL model with appropriate jamming patterns that faithfully represents real conditions.

Using these, we have carried out several simulation experiments to determine the impact of load balancing and jamming on bit throughput.

### **D. ORGANIZATION**

This thesis is organized in five chapters. After a brief introduction, the discussion of FDDI LAN interconnection through CDL is provided in Chapter II. Chapter III presents the proposed NI model development in OPNET. The simulation results are supplied in Chapter IV. The conclusions and recommendations for future studies are in Chapter V.

## **II. INTERCONNECTION OF FDDI LOCAL AREA NETWORKS THROUGH CDL**

### **A. OVERVIEW**

This chapter addresses the issues related to the interconnection of FDDI LANs through the unique environment of the CDL. First, architectural alternatives available for implementing such a connectivity are discussed. Then, existing bridging mechanisms are considered to determine the ideal candidate for a NI for such deployment. The system specific features of CDL are also stated briefly in order to justify the proposed NI architecture.

### **B. ISSUES IN HIGH SPEED LAN INTERCONNECTION**

#### **1. Preliminary**

Local area networks are limited in geography, traffic handling capability, and the number of stations. A single LAN is not likely to meet the needs of many organizations, specifically military applications which require sophisticated command, control and communication functions.

This makes the integration of any LAN topology with other LANs through a communication link necessary. The protocols used to achieve this integration can be at either the network or data link layers of ISO's Open Systems Interconnection (OSI) Model.

Interconnection at the network layer is achieved through routing devices. At the data link layer it is achieved using bridges. Figure 1 shows the layer hierarchies. In general, each layer of an architecture is associated with an additional

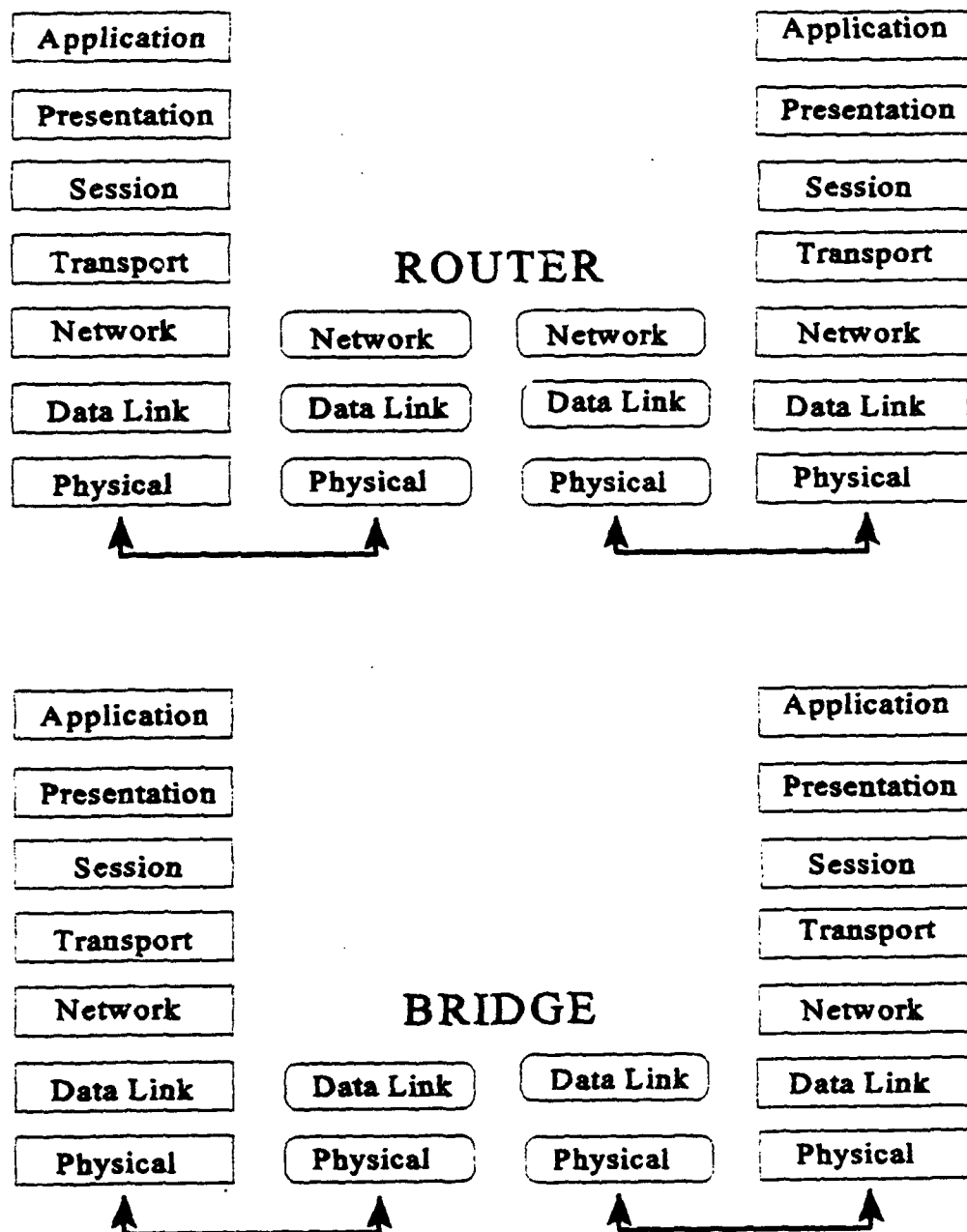


Figure 1: Protocols relayed by the bridges and routers.

level of addressing regardless of the functions performed within a layer. Furthermore, functional distinctions between them are major factors to determine the appropriate layer of interconnection.

## **2. Bridges versus Routers**

Bridges are Medium Access Control (MAC) level store and forward devices that are independent of higher level protocols. They are transparent to communicating end stations. This is particularly desirable in many high speed applications. Bridges make simple forwarding or filtering decisions, based on addressing according to the standardized spanning tree algorithm [2]. This algorithm calls for the automatic discovery of topology changes in the whole network environment.

In a general bridge operation, frames are first received and then conditionally passed to the other LAN depending on a forwarding decision. Effectively, frames are filtered through the bridge. Thus, the frames having destination and source addresses in the same LAN are not forwarded. Although two LANs are connected in the frame level, they do not share each other's local traffic, but only the cross traffic. The purpose of the bridge is to allow hosts attached to other LANs to communicate as if they were in the same LAN.

In contrast to bridges, routers are network layer store and forward devices that rely on an entire higher level protocol suite, and they are explicitly addressed by the communicating end stations [3]. They are capable of searching alternate routes having lower transmission delays and using the best path between two nodes in the network. Since they can tolerate failures in links and stations, routers are designed to enhance availability through the entire network with the penalty of additional processing overheads introduced by the network layer. Therefore, bridges become more important in a high speed network environment.

Recently, both technologies have begun to converge such that simultaneous operation of MAC level bridging and multiple protocol routing services are provided in a simple device. These devices are called as *brouters* [4]. Consequently, the individual attractive features of both architectures are supported with the brouters.

In this study, the NI developed for an FDDI LAN interconnection is intended to be a generic model. Other high speed interface models can be developed either for ATM or newer networking technologies, as they reach maturity.

### **C. PROPOSED BRIDGING METHOD**

#### **1. Current Standards**

Throughout the networking evolution, bridging standardization procedures addressed the interconnection of separate LANs at geographically close points of attachment with local MAC bridges [2]. As LANs became more prevalent both in commercial and military applications, the need to interconnect geographically separated LANs proved inevitable.

An emerging IEEE standard is being developed to satisfy this fundamental necessity [5]. This draft standard specifies how a cluster of remote LANs can be bridged with remote bridges in the MAC level. Moreover, the proposed standard provides all functionalities of local MAC bridges to remote MAC bridges with special additions.

#### **2. Remote Bridging**

Remote MAC bridges, possibly not constrained to the same geographic area, interconnect locally situated LANs to the one or more remote MAC bridges by using a non-LAN communication medium as depicted in Figure 2. MAC service support of remote bridges performs the equivalent communication functions of the conventional local bridges across a non-LAN medium. This communication medium,



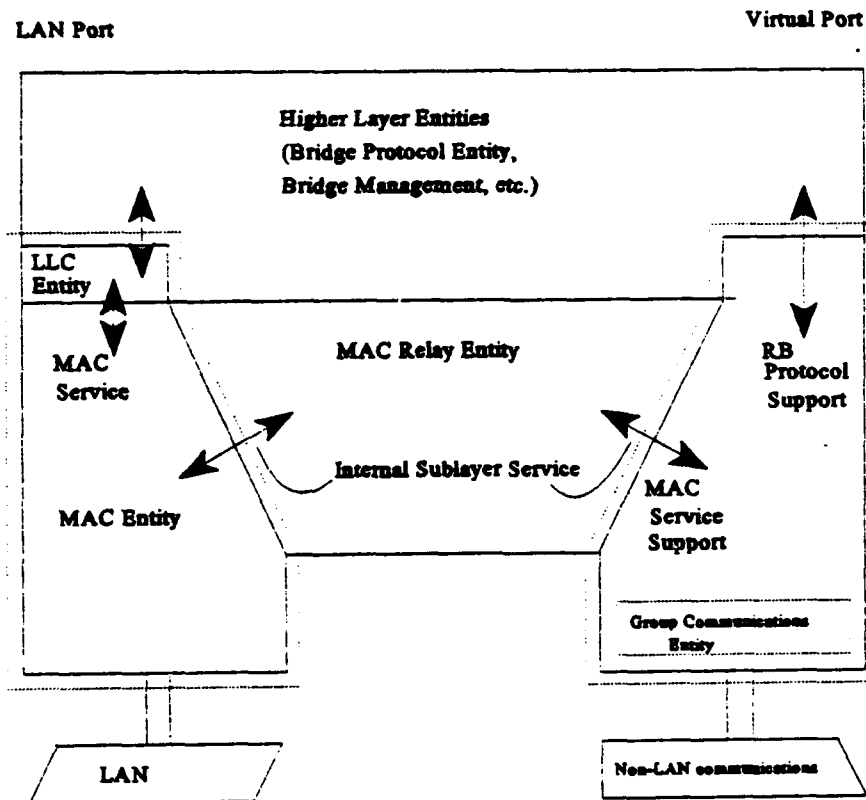


Figure 2: Remote MAC bridge architecture [5].

which may be any type of point-to-point link, is operated and controlled by the group communications entity within the remote MAC bridge.

The group communications entity, which is represented in the virtual port level, is an abstract communication functionality supported by protocols and procedures. As will be stated in the next section, a promising candidate protocol for CDL deployment is the Internet Standard Point-to-Point Protocol (PPP) and its imbedding into this entity is being addressed in further studies [6].

The spanning tree algorithm and its associated procedures are reemphasized in the draft standard for remote MAC bridges in order to preserve the active topology and provide automatic reconfiguration in the entire network.

Remote bridge functions enforced by the spanning tree topology also include forwarding process and learning process associated with filtering database as in the case of local MAC bridges. The forwarding process filters the frames coming from local LAN on the basis of permissible destination addresses contained in the filtering database, while forwarding the frames coming from other LANs to their destinations. The learning process updates the filtering database by observing the source addresses of received frames.

The developed model that will be discussed in the next chapter is evaluated in relation to the proposed remote MAC bridging functionality. However, it is not intended to be the implementation of a full-fledged remote MAC bridge. Since topology is restricted to a pair of FDDI LANs and probable topology changes are not modeled in this generic implementation, bridge functions are fitted into two separate stations existing in the interconnected network model. As a result, subsequent processes for these functions are preferred to be simple forward and filter decisions based on *a priori* knowledge of the addressing database maintained in these stations.

## D. UNIQUENESS OF THE CDL ENVIRONMENT

### 1. Description

As noted before, CDL is a full-duplex, jam-resistant, point-to-point microwave communication system that provides real-time connectivity and interoperability between multiple collection platforms and surface terminals. In our study, CDL is assumed to be a bundle of unidirectional point-to-point bit pipes associated with certain data rates along the communication medium. The bit pipes carrying

information from surface terminals to the collection platform are grouped together as the *command link*. The link which performs the same functionality in the opposite direction is called the *return link*. The return link carries information such as voice, platform status, and data gathered by sensors from the collection platform.

## **2. System Requirements**

The first set of requirements is simply related to NI architecture and its operation. Our model assumes the presence of FDDI LANs on both ends of the CDL link itself. In this thesis, the NI functions required when CDL link is established are addressed. Thus, the setup procedures such as link establishment and authentication are ignored in order to simplify the model simulation sequence. The second set of requirements refers to the typical CDL scenarios. The NI implementation should be generic so that different scenarios can be supported with a single type of NI. Some of the scenarios may require the LAN to be interconnected to multiple LANs permitting multicasting of collected data while others may require relaying of data from one LAN to another. The third set of requirements determines the issues related to data types and quality of service (QoS). QoS, associated with the link, is expected to be provided in the NI using different buffering schemes. Dynamic monitoring of link quality will be deployed as a subsequent subprotocol of PPP in the further developments of our model.

## **3. Asymmetry between Command and Return Link**

The number of bit pipes, and consequently data rate of the return link depend upon the particular CDL configuration used. The information is assumed to be multiplexed, formatted, and transmitted as a composite stream to the surface LAN at various data rates from 10 Mbps to hundreds of Mbps in the full-capacity configuration. Several channel hierarchies are also assumed to be available for each data rate with each channel being an independent data stream.

As previously stated, the command link handles the transmission of user commands from the surface LAN to the collection platform LAN. Contrary to the return link, the command link is assumed to employ a fixed data rate of 200 Kbps. The asymmetry cited in the duplex point-to-point link, enforces separate performance evaluation of the links that will be examined in the Chapter IV.

#### **4. Considerations on Link Utilization**

The major issue to be addressed in this unique network architecture is efficient utilization of the communication link. In CDL deployment, the link is time-critical, bandwidth is expensive, and prone to errors due to the probable interference and jamming in the operational environment.

The implementation of an interconnection mechanism as multiple channels for the return link requires proper management of these channels. There are several studies about high speed protocol controllers which investigate and propose high performance [7-9].

One of these approaches is the realization of a distributed multilink system which offers a load balancing mechanism for protocol controllers in order to increase the total throughput of a high speed data link control system. The proposed method for distribution of frames to multiple links evaluates several transmission allocation algorithms. Our model employs two of the most efficient algorithms which are implemented in the NI. First, we modeled the circular allocation algorithm, which calls for the allocation of frames to the NI transmitters in a circular order regardless of the state of individual transmitter buffers [7]. Secondly, we implemented the empty selection algorithm, which requires the selection of an empty area in transmitting-waiting buffers for frame allocation [7]. The decision process, based on determining the next candidate for transmission allocation, is also carried out by NI in order to provide transparency to end stations.

Although load balancing is aimed towards the efficient multiple link utilization, side effects introduced by this procedure must be examined carefully. The slippage among multiple links due to different transmission capacities causes non-deterministic arrival times of data on the receiving end. For this reason, after the frames are received, reordering of them is unavoidable. Thus, the trade-off is increased receive-end buffer size necessary for this process or efficient multiple link utilization. Our model does not address the issues about resequencing of frames on the receiving end. Instead, a simple time division multiplexing procedure is performed for transmission such that consecutive frames coming from the same station are sent through the same transmission channel regardless of the load balancing algorithm in use. Thus, the reordering problem is solved automatically with this simplification.

## **5. The Effects of Jamming**

Another feature intrinsic to CDL is varying bit error rate (BER) affecting links severely due to the nature of the deployment environment. A constant BER can not be assumed in the model during the existence of the link. The exact system performance must be evaluated in the presence of jamming for any typical military application. Therefore, the link efficiency must be investigated under two different types of jamming models which are described in the next chapter.



### **III. MODELING CDL NETWORK INTERFACE IN OPNET**

#### **A. OVERVIEW**

As discussed in Chapter II, the necessity to integrate two FDDI LANs to each other in CDL scenario, enforces crucial changes in the modular structure of OPNET model. All of these changes are implemented in three phases:

- Applying the model modifications necessary to implement individual traffic, throughput etc. for separate FDDI LANs,
- Modeling the linking nodes in order to accommodate the generic remote MAC bridge features in relation to CDL, and
- Linking two FDDI LANs in CDL perspective.

This chapter provides the detailed model development in the order above. The explanation of FDDI protocol and model in OPNET will not be presented here.

The whole documentation for simulation development is available in the eleven volume set of manuals provided by MIL 3, Inc. In addition to these manuals, [1] serves as an extensive tutorial, particularly, for FDDI LAN development. The experiences gained through previous studies are readily documented in the aforementioned thesis.

The faithfulness of the developed model will also be discussed in the last section of the chapter.

## **B. REVISED FDDI LAN MODEL**

### **1. FDDI LAN Model in Brief**

OPNET provides a built-in model for FDDI LANs. The modifications achieved in a recent study for model development are focused on validating the FDDI protocol in OPNET [1]. It includes the synchronous and asynchronous transmission characteristics of the model. The individual throughput, mean delay and end-to-end delay features for the synchronous, prioritized asynchronous and total traffic in an FDDI LAN were modeled and examined. FDDI multicasting capability and a rudimentary linking node for the interface development are also introduced in [1].

OPNET's FDDI LAN model is implemented in a hierarchy. Each LAN is represented as a ring of FDDI stations connected to each other. Figure 3 shows the complete ring representation of an FDDI LAN. The internal structure of a 10-station FDDI LAN ring is depicted in Figure 4.

Each station is represented as an FDDI station model which includes nodes connected to each other. An FDDI station model is illustrated in Figure 5. Each node is defined by process models that are represented as state transition diagrams. Figures 6-8 show this modular model structure. The detailed explanations of the models will not be reviewed here for the reasons mentioned before. A brief introduction to revised FDDI model in OPNET is intended to lead to a better exposition of the model enhancement made in this thesis.

The ultimate source that determines the behavior of a state in a process model, is "C" language codes embedded in that state. Thus, the modifications implemented throughout this thesis refer to code changes as well as the illustration of interconnected FDDI LAN model.





ring0

Figure 3: FDDI LAN ring representation.

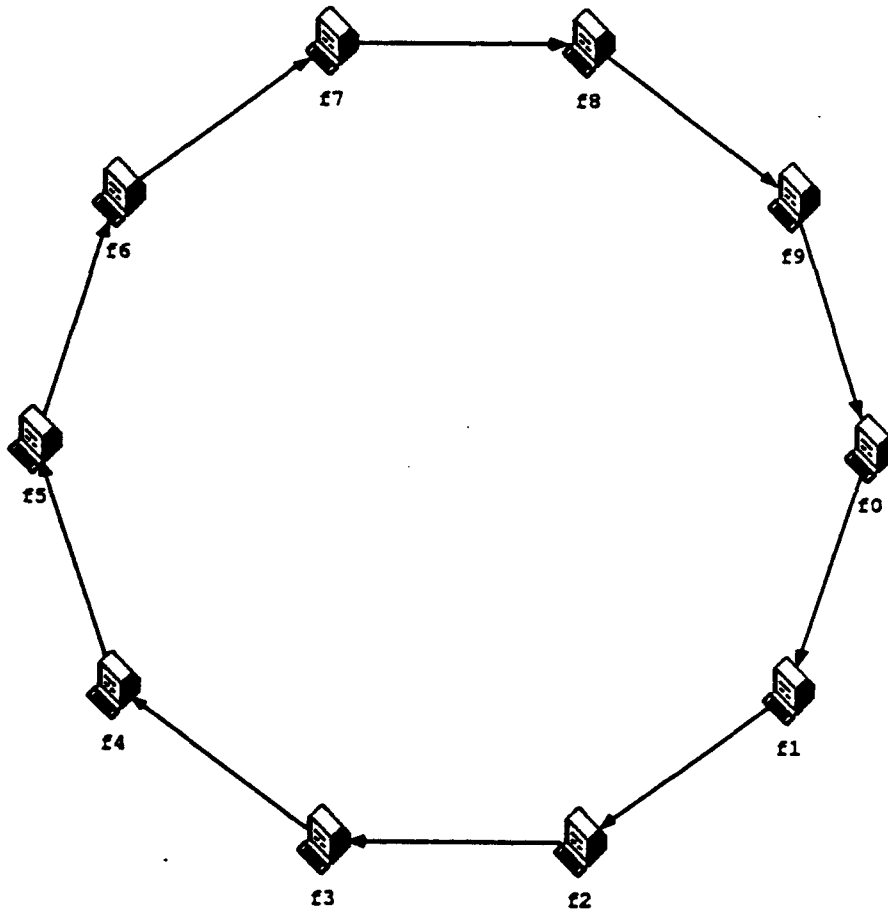


Figure 4: Ten-station FDDI LAN, fddi\_net\_10.

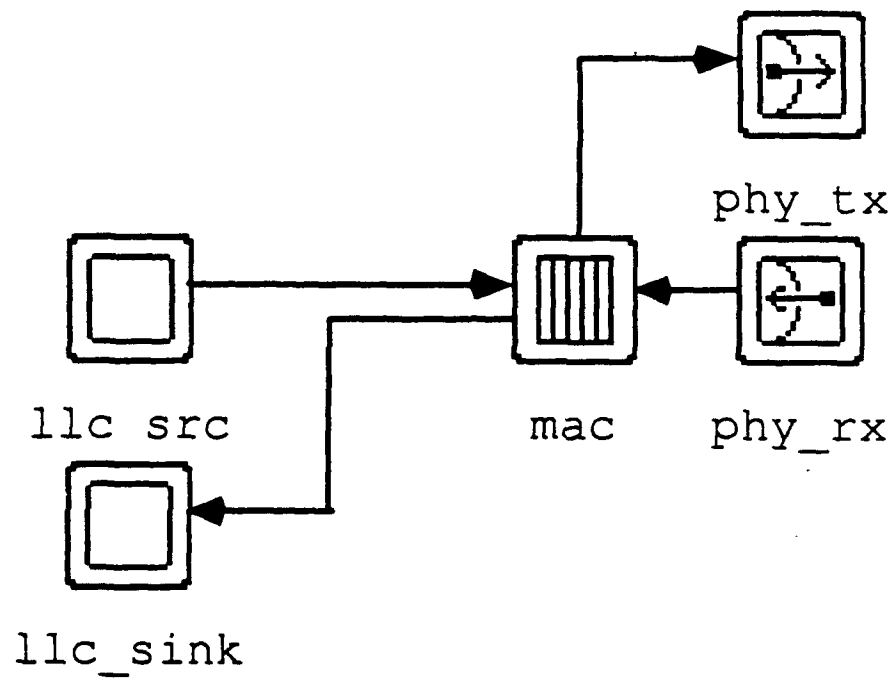


Figure 5: FDDI station model, `fddi_station`.

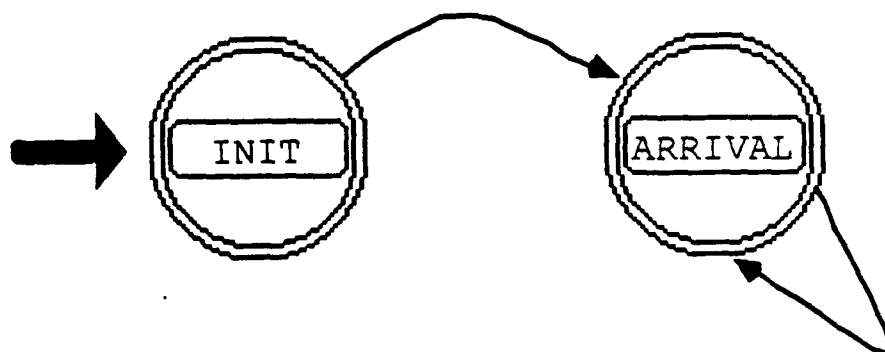


Figure 6: Source process model, `"fddi_gen"`.

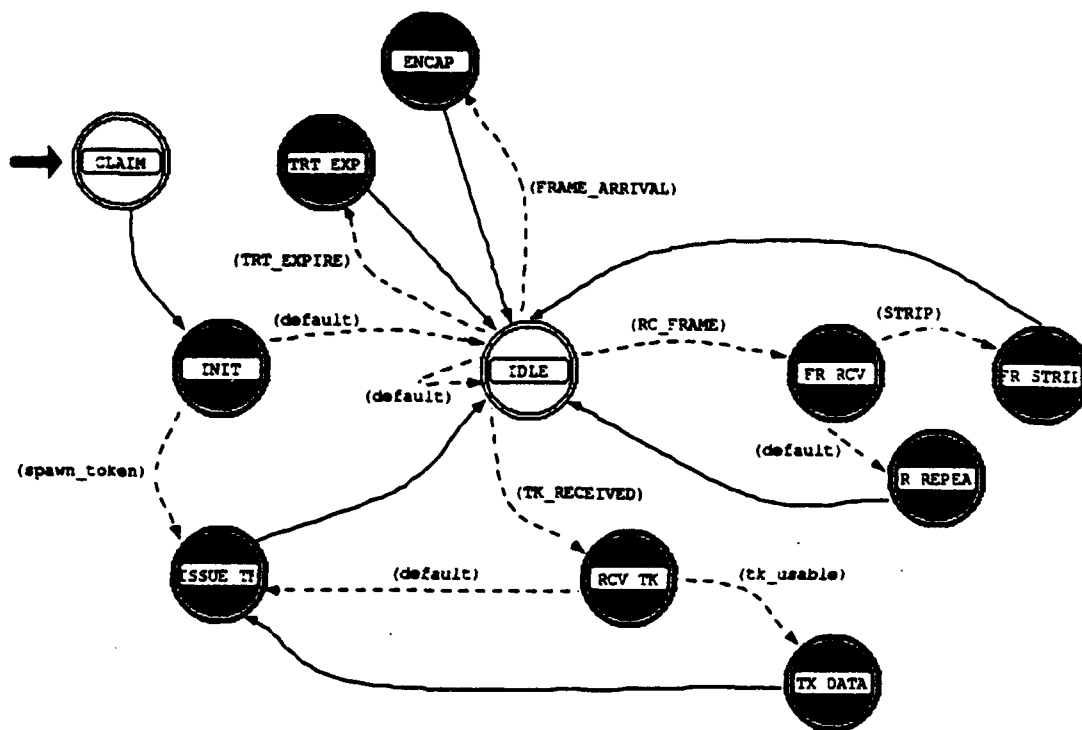


Figure 7: MAC process model, "fddi\_mac".

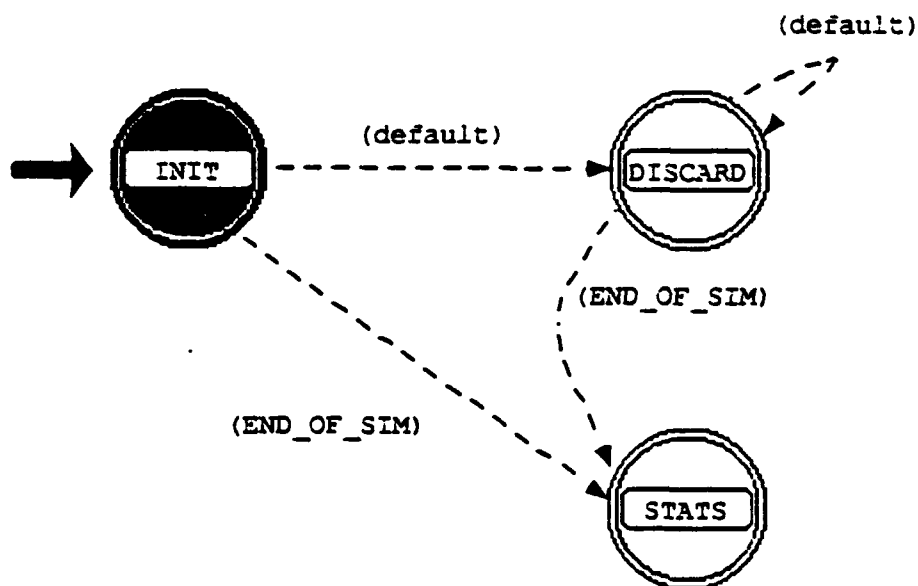


Figure 8: Sink process model, "fddi\_sink".

## 2. Modifications on FDDI Station Model

### a. Preliminary

The built-in FDDI LAN model in OPNET is designed such that only a single LAN's statistic can be obtained at the end of the model simulation. This restriction forces revision of the FDDI station model, and subsequently, its associated process models.

Some of the packet formats, used by all FDDI stations, are also modified in order to enhance the functionality of these models in relation to the interconnection.

Throughout the description of model development, OPNET objects are highlighted with a typewriter font as a technical convention. While node models are highlighted with a typewriter font, process models and simulation attributes are set off in double quotes with the same font. Similarly, packet formats are highlighted in italics and double quotes within the text.

### b. Source Node Modifications

The message traffic, in form of packets is generated in `llc_src` which employs the "`fddi_gen`" process model. The information packet generated to be sent to `mac` is defined as "*fddi\_llc\_fr*". This packet format is also same for the packets sent from `mac` to `llc_sink`. Moreover, the necessary interface between `llc_src` and `mac` is provided by "*fddi\_mac\_req*" interface control information (ICI) packet format. After an information packet is received in `mac`, "*fddi\_llc\_fr*" changes its form to "*fddi\_mac\_fr*". Regardless of the functional differences between them, all of these packets are modified in order to be used in a bridge model. Consequently, three lines of code is added in the ARRIVAL state of "`fddi_gen`" process model to set the new fields of "*fddi\_llc\_fr*" and "*fddi\_mac\_req*" packet formats.

The details about the modifications on packet formats will be stated in Section E of this chapter.

### **c. Sink Node Modifications**

The packets are counted and the statistics are gathered in the `llc_sink`, or in other words "`fddi_sink`" process model. All statistics related to a LAN are updated through the global variables in this process model. In the case of a second LAN, this global nature must be restricted to individual LANs to provide a realistic representation of the whole network. Thus, all statistical attributes in INIT and STATS are renamed to make them private for the first FDDI LAN. A similar procedure is performed again for a second "`fddi_sink`" process model. This process model is named as "`fddi2_sink`", and it is embedded into `llc_sink` nodes of the second FDDI LAN that will be interconnected. Furthermore, the DISCARD state is modified such that statistics are gathered only for local traffic on the basis of incoming frames' source addresses.

## **C. BRIDGE MODEL**

### **1. Preliminary**

The generic remote MAC bridge that interconnects two FDDI LANs is composed of two separate NIs. Since symmetric modifications are done for both LANs, for simplicity, the NI in the collection platform LAN is referred as CPNI, and the NI in the surface LAN is referred as SPNI in our FDDI-CDL model.

### **2. The CPNI**

#### **a. Station Model**

This represents the FDDI station model functioning as an NI in the collection platform LAN. Figure 9 is an illustration of the CPNI employing

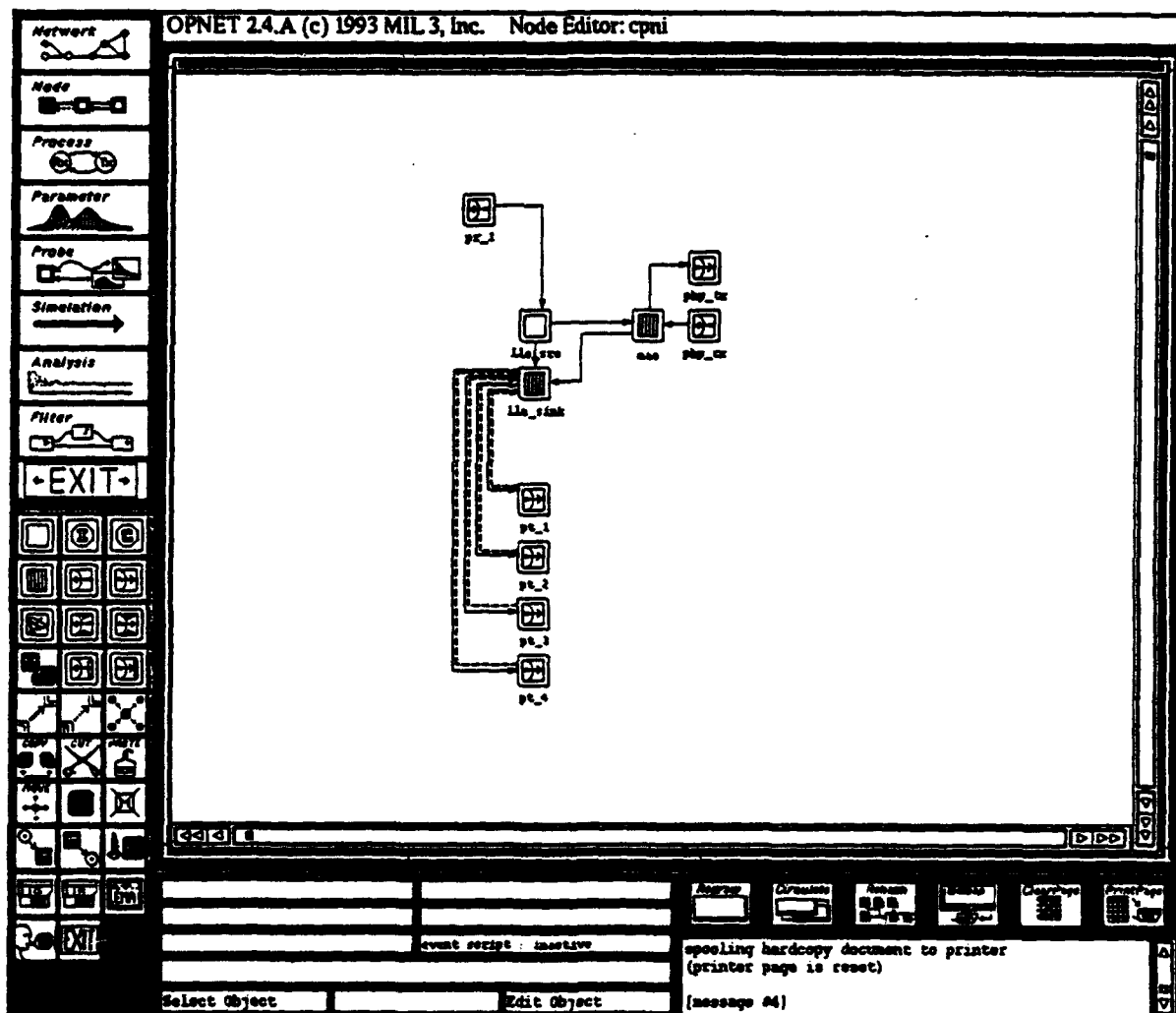


Figure 9: Collection Platform LAN Network Interface (CPNI).

simplified 137.088 Mbps mode return link hierarchy in the OPNET user interface window. When compared to the built-in FDDI station model of Figure 7, first major distinction seen is the implementation of `llc_sink` as a queue module. This alteration is the fundamental step in creation of a bridge link. The other differences from the original model include point-to-point transmitter and receiver nodes.

This particular station examines the destination addresses of frames coming from both LANs. Based on the addressing, frames are either destroyed or forwarded to destination. If the frames are destined for the local LAN, they are simply passed to the local MAC level as in the FDDI protocol. The frames destined for surface LAN are streamed through transmitters' buffers from `llc_sink`. Since the CPNI also has the full functionality of an ordinary FDDI station, any frame sourced from either LAN may be destined for this station address.

While the frames destined to the CPNI coming from its local LAN are treated within `mac`, the overhead of MAC access for the frames coming from remote LAN via `pr_1`, is prevented in Logical Link Control (LLC) level. These frames are evaluated in the `llc_src` and conditionally passed to the `mac` for transmission along the local LAN, or forwarded to `llc_sink` for destruction. Therefore, the frames destined for the CPNI do not need to be repeated. Instead, they are by-passed to `llc_sink` for higher layer's access.

The point-to-point transmitters, named `pt_1` thru `pt_4`, are the return link transmission sources. Information gathered in collection platform LAN is conveyed to the stations on surface LAN by these transmitter nodes. Transmission allocation procedure is realized according to the load balancing algorithm in use.

Conversely, the point-to-point receiver node, `pr_1` is connected to `llc_src` and it serves as the command link gate to the collection platform LAN.

### b. Source Node Modifications

The source node of the CPNI includes "cp\_fddi\_gen" process model. This process model differs from "fddi\_gen" due to the modifications implemented in its ARRIVAL state. In this state, frames coming from the surface LAN are determined first. If there are incoming frames, the CPNI postpones its own frame generation until no more frames are received. Then, the received frames' destination addresses are checked. Any frame destined for the CPNI is simply forwarded to llc\_sink for destruction. Thus, these frames are not sent to mac like the rest of the incoming ones. The necessary interface along with the information packet is supplied to mac with "fddi\_mac\_req" ICI packet format. During this ICI packet transfer, "pri" field is set to the highest priority which is assigned to the synchronous transmission. This procedure is essential to keep the bridge model realistic. In FDDI, these priorities relate to LAN bandwidth allocation. Since priorities are only local to each LAN and FDDI frames do not contain an explicit priority field (unlike IEEE 802.5), there is no way to determine to which priority level the frame received from the other LAN belongs. Therefore, once the NI receives these frames, it always sends them as synchronous traffic on its LAN. The ".C" code for "cp\_fddi\_gen" is provided in Appendix A.

### c. MAC Node Modifications

The mac of the CPNI employs "cp\_fddi\_mac" as its process model (Appendix B). The modifications made to the original model are categorized in four groups as below.

(1) "static" Declarations. Since we use common basic process models in stations with different names, the functions and variables used by the process models are made 'static' to prevent name conflicts.



(2) **INIT State.** The token can be generated by any of the stations in the original OPNET FDDI model. To simplify the simulation sequence, the INIT state code is changed such that only the NI is capable of generating the first token. This takes effect only if the simulation environment file is set as described in Chapter IV.

(3) **FR\_REPEAT State.** The frames coming from the physical layer are inspected on the basis of destination addresses in this state. The modifications in FR\_REPEAT refer to the implementation of major bridge functions. While the frames addressed to the surface LAN are forwarded to `llc_sink`, the ones having a local destination address are propagated in the local LAN. Thus, the local traffic is effectively filtered by the NI.

(4) **ENCAP State.** The original source address of a frame coming from `llc_src` needs to be preserved in this state. Furthermore, a simple check is made for default values of source and destination addresses in "`fddi_mac_req`" ICI packet format. Thus, "`fddi_mac_fr`" packets containing the same source and destination addresses are not erroneously composed in this state.

#### d. Sink Node Modifications

Several modifications are required to the `llc_sink` of the NI as described below. This node uses the "`cp_fddi_sink`" process model and acts as an interface between the FDDI LAN and CDL. As a result, all three states of this process model are modified. The modified ".C" code is in Appendix C.

(1) **INIT State.** All the global statistics' arrays which are used in the analysis tool are defined here. Consequently, the statistics array needed for the traffic monitoring of return link in each priority, is also defined in this state.

(2) **DISCARD State.** Notwithstanding its name, the "`cp_fddi_sink`" process model's DISCARD state does not discard all the frames.

The source and destination addresses of the frames are inspected here. The frames having the NI address as their destination are destroyed. Moreover, if they are generated by local stations, the related statistics are updated. The frames destined for the remote LAN are not destroyed. Instead, they are counted and enqueued in the subqueues of `llc_sink` for transmission. The transmitters' status are continuously monitored before and after the transmission with OPNET statistical interrupts described in Section D. The allocation of frames to transmitters using different load balancing algorithms is also performed in this state. The details of this procedure are provided in Section E.

(3) **STATS State.** This state refers to the documentation of statistics at the end of the simulation. It is this state that must generate the return link related statistics.

### 3. The SPNI

#### a. Station Model

This model is one of the FDDI stations located in the surface LAN. The SPNI employing the same return link hierarchy mentioned above, is depicted in Figure 10. This station has the same functionality of the NI as in collection platform LAN, but in the reverse direction. As clearly seen, there exists symmetry in the number of transmitters and receivers with respect to its correspondent in collection platform LAN. The node, `llc_src` accesses `mac` via `pr_1` thru `pr_4` that are the downstream gates of surface LAN. The frames destined for the SPNI are distinguished by `llc_src` and forwarded to the `llc_sink` directly. This process prevents the additional MAC access creating a significant overhead.

Although the SPNI serves as a receive end-point for the return link, it is also capable of generating its own frames like other FDDI stations.

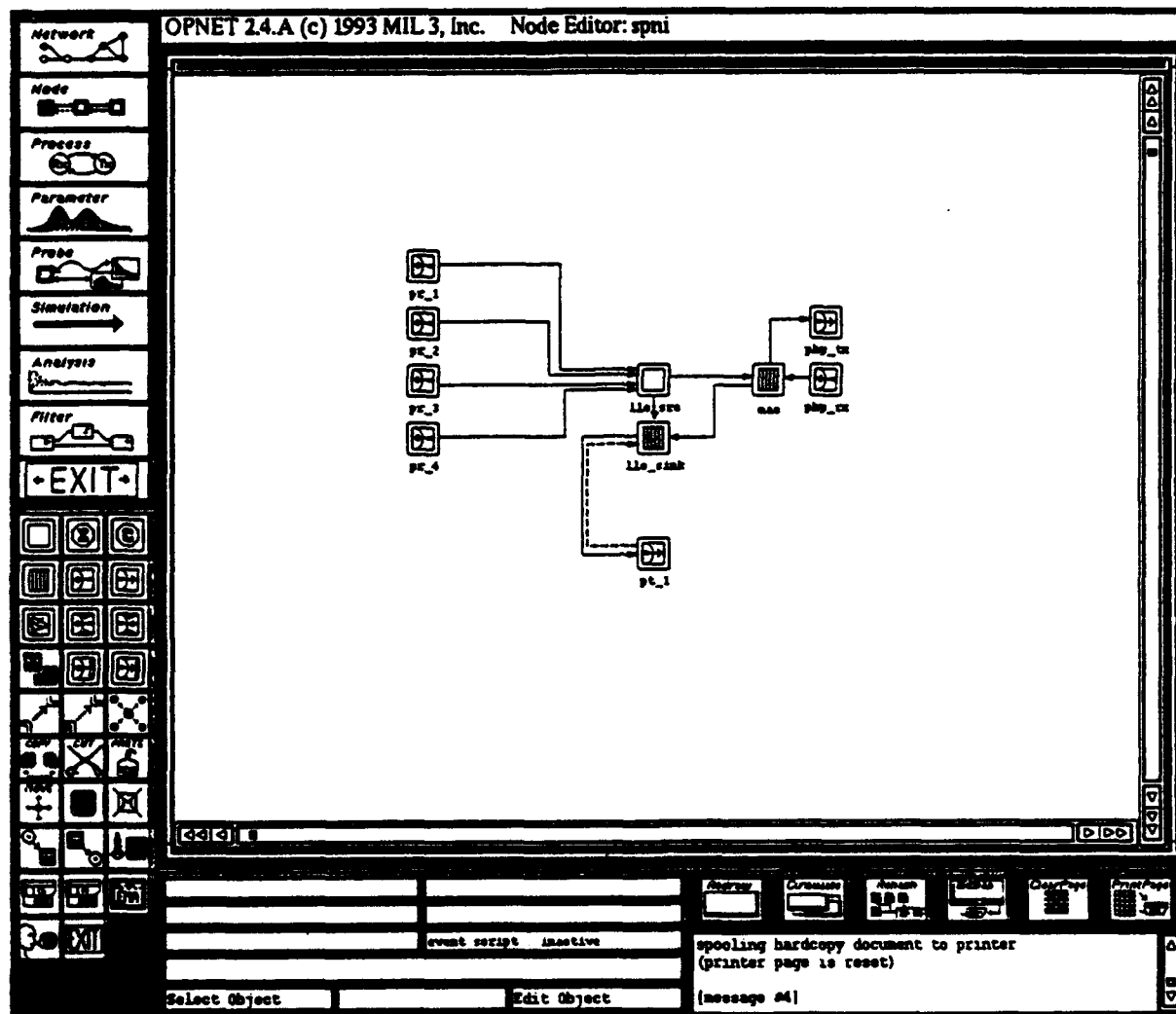


Figure 10: Surface Platform LAN Network Interface (SPNI).

The transmitter, `pt_1` is command link access node. This node handles the frames coming from `llc_sink` and delivers them to the collection platform LAN. The forwarding and filtering decisions based on addressing, are implemented in `llc_sink` as in the CPNI.

Symmetric modifications are carried out in the process models of `spni` with respect to the CPNI. Since much of the details are provided before, only major distinctions for each node will be stated below.

#### **b. Source Node Modifications**

The source node of the SPNI employs "`sp_fddi_gen`" as the process model. The modifications implemented in this process model are the same as "`cp_fddi_gen`" with one exception. In the ARRIVAL state, the frames having destination addresses in collection platform LAN are passed to `llc_sink`. This inspection is based on the permanent database maintained in the NIs as can be seen in the ".C" code supplied in Appendix D.

#### **c. MAC Node Modifications**

The process model "`sp_fddi_mac`" has the same alterations as "`cp_fddi_mac`" process model's case in INIT and ENCAP states. The same variables and all functions are also defined as "`static`". The difference appears in FR\_REPEAT state. Thus, ".C" code provided in Appendix E includes only this modified state. In this state, the frames belonging to the collection platform LAN or addressed to the SPNI are passed to `llc_sink`, while others are simply propagated. This decision is also made on the basis of *a priori* knowledge of addresses.

#### **d. Sink Node Modifications**

The changes in INIT and STATS states of "`sp_fddi_sink`" are symmetric to the ones in "`cp_fddi_sink`" process model. Furthermore, DISCARD

state modifications differ from "cp\_fddi\_sink" process model's DISCARD state. In this state, the frames destined to the SPNI are destroyed and related statistics are updated for the surface LAN. Since there is no multiple link deployment for the command link, transmission capacity allocation is based on FIFO queue of llc\_sink. For this reason, load balancing algorithms are not employed here. The ".C" code for "sp\_fddi\_sink" is provided in Appendix F.

## **D. CDL MODEL**

### **1. OPNET Model for Point-to-Point Links**

#### **a. Preliminary**

OPNET system models are composed of distributed subsystems which need a communication mechanism among them. There are several methods available in OPNET for the communication between two subsystems, the most prevalent being the packet-based one. In the OPNET environment, a packet is a data structure facilitating information transfer from one subsystem to another. While packet streams, represented as the physical connections, provide transmission between nodes in the same subsystem, the ultimate transfer of information to other subsystems is employed in the form of communication links.

As mentioned in Chapter II, CDL deployment requires point-to-point link management because of the physical constraints of the environment. Point-to-point links, either simplex or duplex, allow packets to be transmitted between a single pair of nodes. Each link consists of several transmission channels between the source and destination node that it connects. In OPNET, these nodes are referred to as transmitters and receivers.

#### **b. Transmitters**

These nodes serve as the exit points of a station for packets forwarded on point-to-point data transmission links. They are composed of multiple channels, each of which is tied to a receiver channel in a remote station via point-to-point link.

Transmitters are built-in FIFO queues with infinite capacity per channel. These queues regulate the transmissions in a channel so that only one packet is transmitted on the link at any time.

The status of any transmitter can be monitored with OPNET statistical interrupts. These interrupts are represented via statistics wires in the OPNET user interface window. Particularly, the "busy" statistic, which is fed back to a node from a transmitter plays the most important role for any transmission capacity allocation process.

#### **c. Receivers**

As opposed to transmitters, these nodes act as entry points of a station for packets received on point-to-point data transmission links. They employ the reverse functionality of the transmitters. After reception of packets on channels in the remote station, the packets are forwarded through output streams to the attached node of the receiver node for further processing.

#### **d. Transceiver Pipeline Stages**

In OPNET, point-to-point links can be configured to model packet transmission in several ways. Each link includes a series of default or user-definable submodels called pipeline stages. The source code for all the default models is provided in `<opdir>/stdmod/base` directory.

In any pipeline stage of the model, the data related to each packet is used in the various computations related to its transmission. These computations are performed in order to identify the reception time and whether or not a packet is received correctly.

Point-to-point links are based on a four stage pipeline that supports the transfer of packets from a transmitter to a receiver. These stages are described below in brief.

(1) **Transmission Delay.** This is the first stage of the transceiver pipeline. In this stage, the amount of time required for the transmission of an entire packet is calculated. This computation is performed independently for each packet transmission on the basis of channel “data rate” and length of the packet. OPNET employs the default transmission delay model, “dpt\_txdel” for this pipeline stage.

(2) **Propagation Delay.** After the first stage, packets are passed to the propagation delay pipeline stage. The purpose of this stage is to calculate the amount of time for the packet to reach the receiver in the destination node. The parameter necessary for this computation is the “delay” attribute of the point-to-point link. The default propagation delay model provided by OPNET is called “dpt\_propdel”.

(3) **Error Allocation.** Since the packets are prone to errors during transmission, the third stage of the transceiver pipeline allocates errors for transmitted packets. The default model, “dpt\_error”, uses the “ber” attribute of the point-to-point link for this process. The algorithm implemented in this model generates a random number of errors with a fixed BER during the whole period of the simulation.

(4) **Error Detection and Correction.** This is the final stage of point-to-point transceiver pipeline. The purpose of this stage is to determine

whether or not the received packet can be accepted and forwarded to its destination. The comparison is based on the "ecc" attribute of the receiver node. This attribute represents the probability of bit error that can be tolerated for acceptability. If it is set to zero, default model "dpt\_ecc" only accepts error-free packets. Setting a non-zero threshold for this attribute corresponds to error correction procedure for the received packet.

## **2. Error Modeling over Multiple Links**

### **a. Modifications on Error Allocation Pipeline Stage**

As stated earlier, error modeling over point-to-point links is carried out in the error allocation stage of transceiver pipeline. Since the default pipeline stage model uses a constant BER as the simulation progresses, a realistic model must be developed for the CDL link. For this purpose, default error model "dpt\_error" is modified, and the new model is renamed as "cdl\_pt\_error".

In OPNET, the necessary modifications for a pipeline stage model are done in the "*op\_models*" directory. And, after these modifications, the customized model file must be compiled separately. So, "cdl\_pt\_error" model is compiled with the following command:

```
cc -c ~/op_models/cdl_pt_error.ps.c -I/<opdir>/sys/include
```

This procedure is necessary to link other OPNET libraries. Otherwise, binding errors result during the generation of simulation file. Appendix G is the file "cdl\_pt\_error.ps.c", containing the modifications described in this section.

In contrast to the default model, "cdl\_pt\_error" performs error allocation with a varying BER. Thus, the link attribute "ber" is disregarded in this stage. Instead, new attributes are specified to accomplish a dynamic error allocation



stage. Six different attributes are added for each channel in the extended attributes menu of the point-to-point link. These attributes are depicted in Figure 11, and defined below.

(1) **"jam\_ber"**. This attribute is the maximum BER on the transmission channel of a point-to-point link during jamming.

(2) **"ber\_bet\_jam\_len"**. This attribute corresponds to the maximum BER on the transmission channel of a point-to-point link in the duration between two consecutive jamming pulses.

(3) **"jam\_length"**. This is the duration of a jamming pulse affecting the transmission channel of a point-to-point link.

(4) **"interval\_bet\_jam\_len"**. This is the duration between two consecutive jamming pulses affecting the transmission channel of a point-to-point link.

(5) **"init\_jam\_offset"**. This attribute is the initial offset time on the transmission channel of a point-to-point link. This offset is required for the channel-swept jammer which will be described later.

(6) **"jammer\_type"**. The last one of the extended attributes specifies the type of the jamming model of which CDL link is exposed to.

#### **b. Jammer Implementations**

There are two distinct jamming models implemented in this study. These are pulsed jammer and channel-swept jammer, respectively. Regardless of the jamming type in use, a transmitted packet is first time-stamped in simulation time domain. This time-stamp is used to determine whether or not a packet is subjected to jamming. After this decision, further procedures are carried out for each jammer type independently.

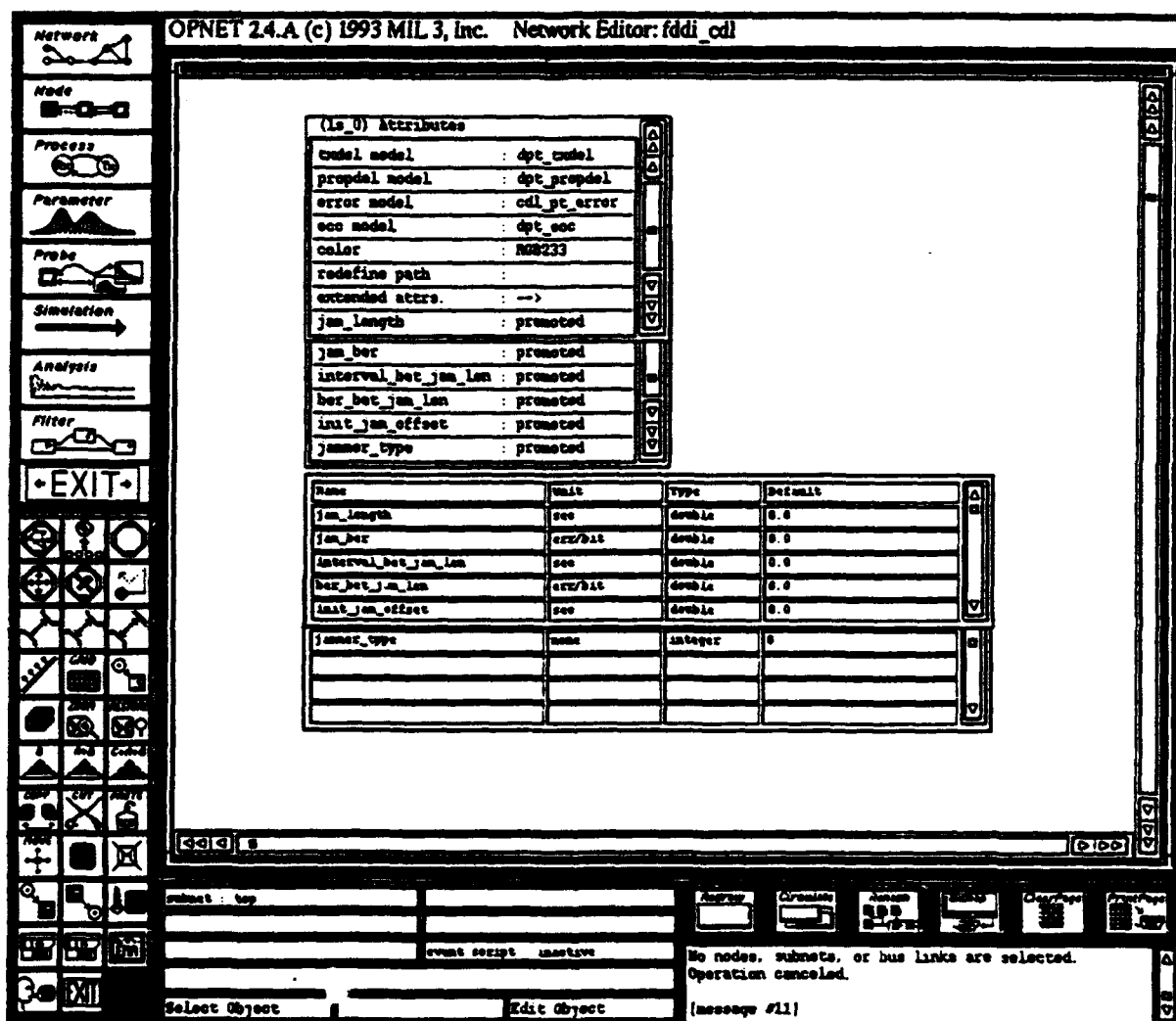


Figure 11: The attributes used for jamming patterns.

(1) **Pulsed Jammer.** In this model, all transmission channels are exposed to separate pulse trains in time. This configuration is achieved by setting up proper values for previously described extended attributes in the simulation environment file. Appendix H is a sample environment file configured for this particular model.

In order to provide a more realistic representation, the durations of jamming pulses and BERs during these periods are also randomized with uniform distribution. Because of this stochastic process, first four of the extended attributes are actually the maximum values that can occur during transmission. Consequently, each transmitted packet is subjected to a different BER according to its presence in simulation time domain. Figure 12 is an illustration of a pulsed jammer.

(2) **Channel-Swept Jammer.** The jammer implemented with this model sweeps each transmission channel consecutively in the simulation time domain. Again, the necessary configuration for this procedure is provided by the extended attributes' values which are specified in the simulation environment file. As opposed to the previous jammer model, the channel-swept jammer does not randomize the durations of jamming pulses to maintain consecutive pulse scheme in order. However, BERs are still randomized with uniform distribution. A proper offset must be specified in the "init\_jam\_offset" extended attribute of the point-to-point link to provide consecutive pulses as depicted in Figure 13.

## **E. LAN INTERCONNECTION**

FDDI-CDL interconnection can be realized using the modifications described so far. Figure 14 shows the ultimate interconnected model. While the top simplex point-to-point link from surface LAN (ring1) to collection platform LAN (ring0) represents the command link, other four links in the reverse direction represent the

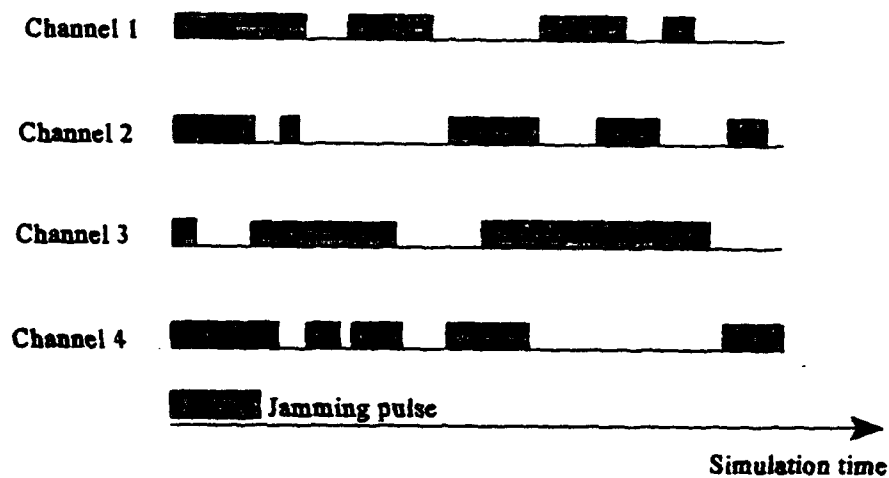


Figure 12: Pulsed jammer representation.

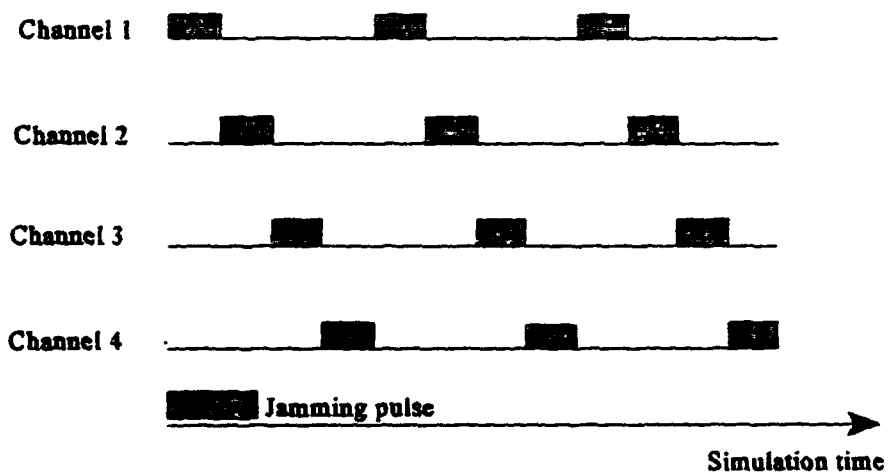


Figure 13: Channel-swept jammer representation.

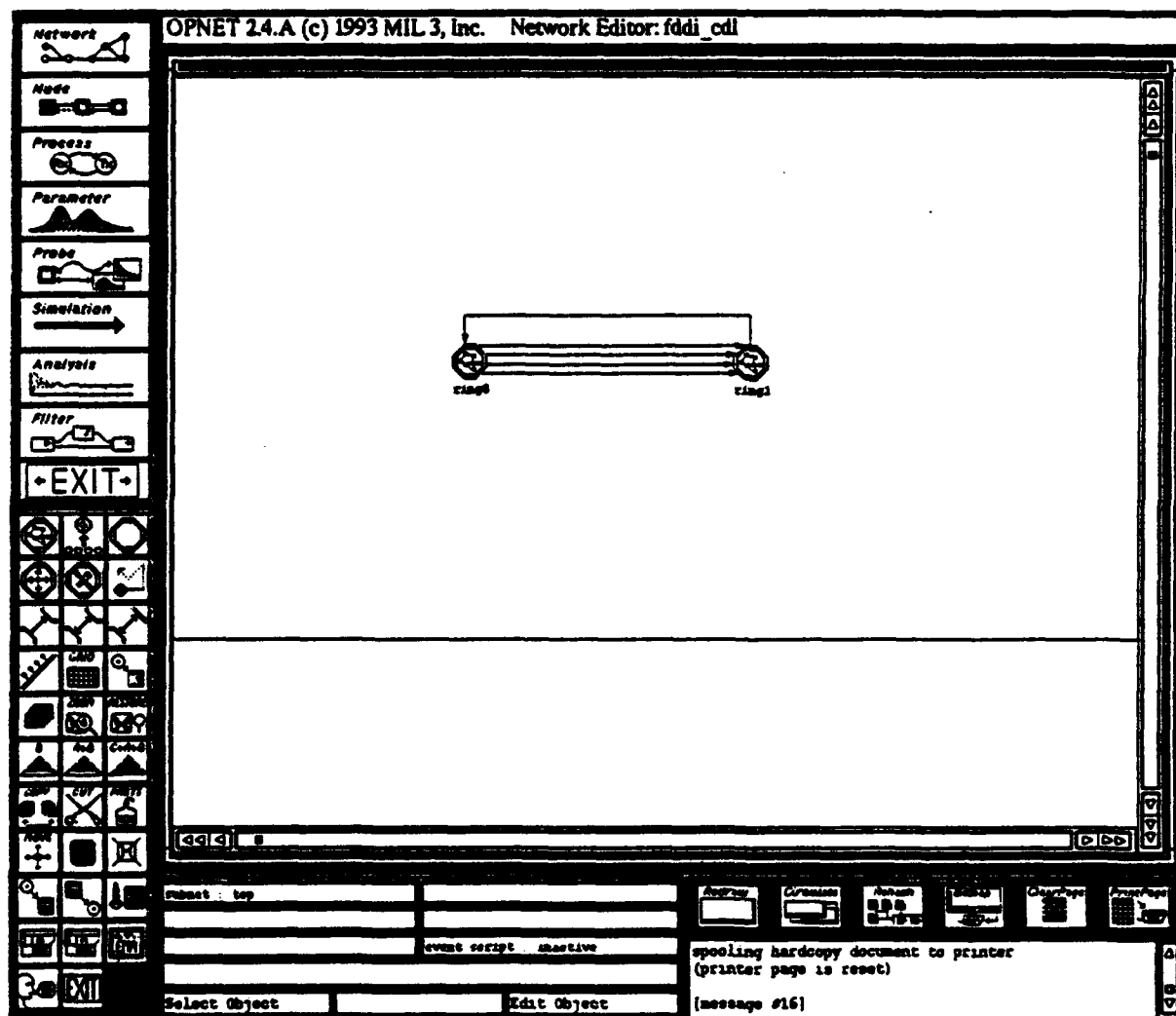


Figure 14: Interconnected network model, fddi\_cdl.

simplified return link hierarchy. In this section, specific features of this model of LAN interconnection are stated.

### 1. Addressing Through The Remote Bridge

The interconnection mechanism is based on the forwarding decision made by the inspection of station addresses. The stations that reside on LANs have unique addresses. Our model employs 10-station FDDI LANs on both sides of the CDL link. In order to simplify MAC bridge functions, these addresses are permanently installed in the filtering databases of the CPNI and the SPNI. These stations, acting as NIs, have the maximum station number of each LAN. Thus, filtering decision is based on the comparison of these NI addresses. If the number of stations or their addresses need to be changed, those NIs' filtering databases must be updated according to the new address assignments. Besides, the DISCARD state of process models of `llc_sink` nodes of the station models in both LANs must also be modified appropriately to obtain individual LAN statistics.

The address information is sent to the nodes with several packet formats in OPNET modeling environment. The packet format, "*fddi\_mac\_fr*", allocates 16 bits to represent station addresses. In current bridging standards [2, 5], the addresses are represented as 48 bits within a frame. Thus, the mentioned packet format is modified to include 48-bit source and destination addresses. Furthermore, the necessity to include source and destination addresses in "*fddi\_llc\_fr*" packet format results in the same modification. This packet format is used in `llc_src` nodes of NIs to prevent unnecessary MAC access as mentioned in Section C.

For interface control purposes, "`src_addr`" field is also added in the original "*fddi\_mac\_req*" ICI packet format. This address is used by the `mac` to keep the original source address of a frame unmodified during the data transfer between

LANs. Figures 15-17 show the resulting packet formats in the parameter editor of OPNET user interface window.

## **2. Load Balancing over Multiple Links**

As stated before, the return link is composed of multiple channels. In our model, simplified return link hierarchy is represented as four simplex point-to-point links, each having one transmission channel. Data rates for each channel on both links are specified as in Figures 18 and 19.

This multiple channel hierarchy requires an algorithm for transmission capacity allocation. The selected algorithm eventually leads to load balancing over multiple links. In the OPNET model implemented, this procedure is carried out by `llc_sink` of the CPNI. The node, `llc_sink`, is in the form of a queue module consisting of four subqueues. Thus, the transmission capacity allocation process is basically allocating frames to these subqueues that are attached to the individual transmitters. In other words, the subqueues act as buffers. Instead of allocating frames to the transmitter buffers directly, this method is chosen to monitor the effects of load balancing algorithm in use. As stated in Section D, point-to-point transmitter buffers are built-in queues with infinite capacity. This approach would not be realistic for a bridge model. In a real bridge, insufficient transmission queue sizes can cause dropping of frames. In the CDL NI model, the frame currently being transmitted resides in the transmitter buffer until its transmission is completed. Other frames waiting for transmission reside in the finite subqueues of `llc_sink`. Then, these frames are forwarded to the related transmitter when the "busy" signal goes low, meaning that the transmitter channel is idle. This is accomplished by OPNET statistical interrupts, and the allocation algorithm calls for the continuous monitoring of this signal.









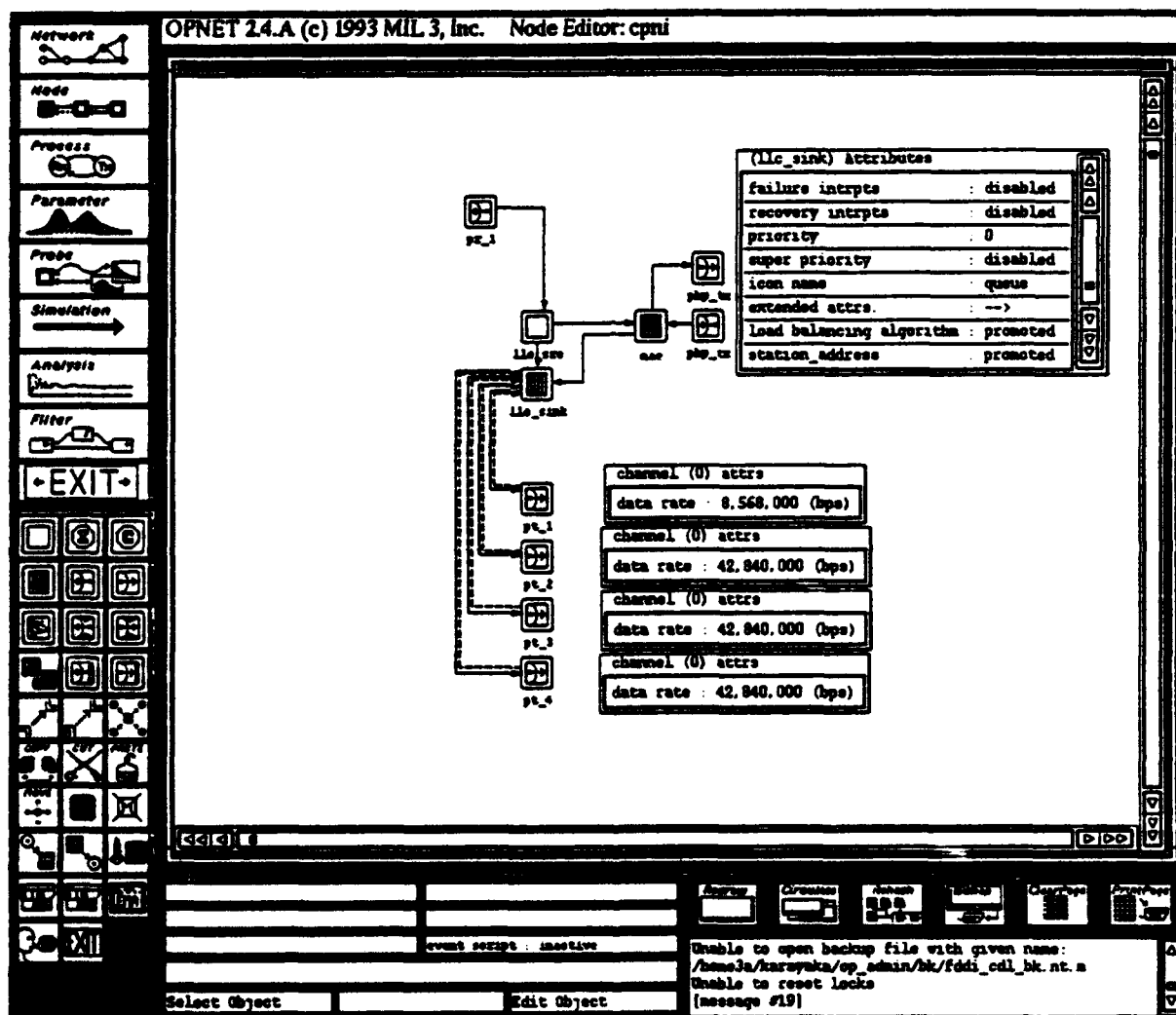


Figure 18: Return link data rates and new attributes for llc\_sink.

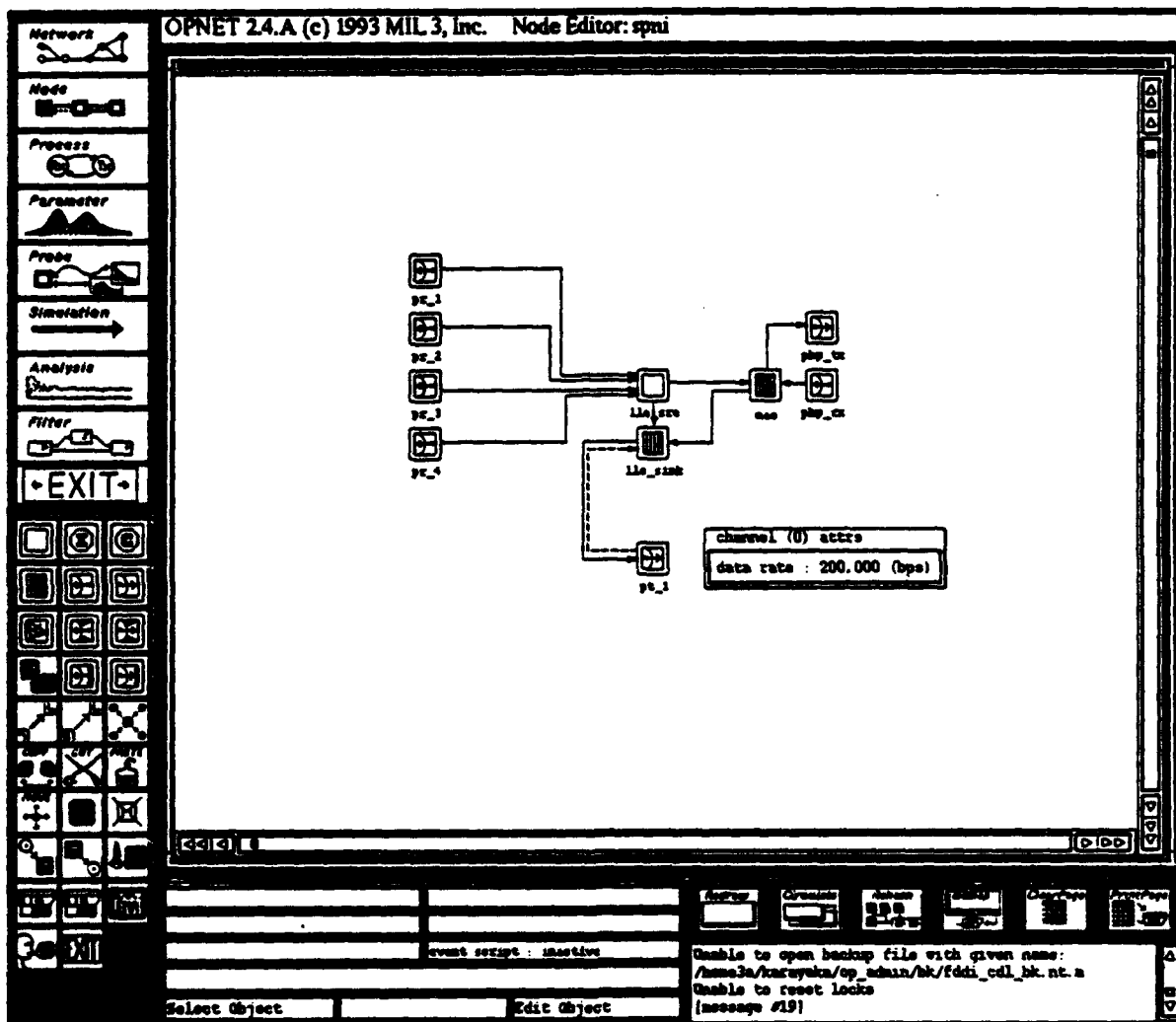


Figure 19: Command link data rate.

The algorithm can be specified for the model from the simulation environment file. So, an extended attribute, "load balancing algorithm", is defined for the CPNI as in Figure 18. Regardless of the load balancing algorithm in use, a frame is allocated to a buffer, each time "cp\_fddi\_sink" process model is executed. Then, the order of subqueues is determined with an incrementing index according to the allocation algorithm employed. Every subqueue has an associated source address field, so that previously allocated frame's source address is maintained in that subqueue. Two algorithms are implemented as described below.

**a. Circular Allocation Algorithm**

Before allocating frames in a circular order to llc\_sink subqueues, the source address of an incoming frame is inspected first. If that frame is sent from a station address which has been previously allocated to that subqueue, the new frame is also buffered in the same subqueue. Thus, consecutive frames from the same source are sent over the same transmission channel. Otherwise, allocation is still done circularly ignoring the state of individual subqueues. Because of the different channel data rates, some buffers can fill up and even lead to frame loss, unless the sizes of these buffers are selected adequately.

**b. Empty Selection Algorithm**

This algorithm refers to the allocation of frames to subqueues having the maximum empty slots. The frequency of the occurrence of empty slots is directly proportional to the data rate of transmission channel. Thus, the buffer sizes needed for this algorithm are likely to be less than the ones needed for the previous algorithm. Allocation of consecutive frames sent from the same station is still implemented in the same manner as the circular allocation algorithm.

## **F. FAITHFULNESS OF THE MODEL**

The CDL NI model is developed with the extensive features provided by OPNET. Modifications to the original models are carried out step-by-step, so that the original modularity is not sacrificed with new enhancements.

The OPNET FDDI protocol model is studied and validated in [1]. Further model improvements of this thesis are realized on the basis of the results of this previous study. During the CDL model development, specific system requirements were primarily considered and adapted to OPNET modeling environment.

In a real CDL deployment, depending on the deployed channel hierarchy, an aggregate bit stream at the high frequency is obtained using multiplexing of the independent transmission channels. Then, at the receiving end, demultiplexing is carried out. Our CDL system implementation does not model the link between the output of the multiplexer and the input of the demultiplexer. Instead, the multiplexer input and the demultiplexer output are modeled and monitored. In other words, the link is not modeled at the aggregate bit stream level, but at the individual channel level. This permits us to view the link as a set of independent channels rather than a single stream of bits without losing the realistic nature of the link.

The corruption of packets in a real jamming environment is of an irregular nature. The bits in a data stream may be inverted, or random sequences of bit patterns may be added to the information packet. In the OPNET modeling environment, error allocation is done by inverting the bits present within a packet. So, the jammer modeling is based on this constraint. In other words we can not model loss of the CDL framing synchronization in OPNET. However, this study is targetted at the data link layer and above. Therefore, this limitation does not present a problem. Despite this limitation, since our jamming models use a varying

BER within the random durations in the simulation time domain, the desired error allocation feature is still provided. In the model, the error allocation process within a link is such that a random number of errors are introduced in a randomly selected packet in each channel. This is essentially what real jamming of the aggregate CDL bitstream will result in. Two types of jamming models are investigated in order to evaluate system performance in a wider variety of scenarios.

Finally, interconnection is realized in terms of an upcoming bridging standard [5], so that dependability of the model is ensured.





## IV. MODEL TESTING

### A. OVERVIEW

This chapter provides several test results in relation to the enhanced capabilities of our model. These tests include the monitoring of traffic in the whole network for individual LANs and the evaluation of transmission capacity allocation algorithms for the return link. The effects of different jamming patterns on the return link are also demonstrated. Similarly, the command link performance is studied in brief.

### B. PERFORMANCE METRICS

During the simulation tests, a moderate traffic load is offered for both LANs that are interconnected. The behavior of the OPNET's FDDI model with varying traffic loads was studied previously [1]. Therefore, the tests are primarily run and evaluated for NI functional characteristics and the CDL link performance during jamming.

The transmission capacity allocation algorithms related to the load balancing over multiple links are monitored in the `llc_sink` node of the CPNI. The buffers within this node are observed in order to evaluate these algorithms' efficiency with the "pksize" and "delay" probes in the OPNET analysis editor. These probes are simply used to determine the advantages and drawbacks of the load balancing implementations. The "pksize" attribute's name is misleading in OPNET probe editor and it refers to the number of packets residing in a subqueue. Thus, while "pksize" probe is the means to determine the required buffer sizes, the transmission waiting periods of the packets in the buffers are monitored with the "delay" probe.

The average BER of the return link is monitored on the SPNI receivers with "avg\_ber" probes for each transmission channel. In this study, the BER of the command link during jamming is not investigated. The utilization of multiple links under different jamming patterns is also examined on the transmit and receive-ends of the link. Results of the different simulation experiments follow in the next section.

## C. TRAFFIC MONITORING

### 1. Overview

The modifications made within this thesis to the OPNET FDDI model provide the individual monitoring of two interconnected LANs. For CDL deployment, the collection platform LAN is likely to direct most of the traffic to the return link, although there may be still an amount of local traffic under consideration. On the other side, the surface LAN directs some of its traffic to the command link while introducing a significant amount of traffic locally. For the reasons specified above, each LAN must be monitored independently to provide a realistic representation of the entire network.

### 2. Setup

The simulation was set up to display local traffic within one LAN and the traffic directed to the remote LAN. Both LANs consist of ten FDDI stations, generating packets at a constant rate. The stations generated 20000-bit packets at an arrival rate of 250 packets per second, so that 50 Mbps of the traffic load is expected for a LAN. The stations f9 and f19 are specified as the CPNI and the SPNI, respectively. While the surface LAN can send packets to all stations in the interconnected network model, the collection platform LAN is designated to direct its total traffic only to the return link. Both LANs are capable of generating 90 percent asynchronous traffic at different priorities. Target Token Rotation Time

(TTRT) was set to 4ms, and necessary synchronous bandwidth was determined as 0.08955675 as the required OPNET parameter. This value is a function of TTRT; it is computed by following the necessary steps as described in [1] and amounts to 0.358227 ms/station.

As mentioned before, the initial tokens are launched by the NIs. Thus, in the simulation environment file, the "spawn station" attribute is disabled by setting it to 20. This number is chosen arbitrarily, but it must be greater than the maximum station number that exists in the entire network model. This is just an OPNET hack to keep the "fddi\_mac" process model of original "fddi\_station" unchanged.

Furthermore, the "accelerate\_token" attribute is also disabled by setting it to zero. When this flag is enabled, the token is blocked in the station which has no packets to transmit. This procedure leads a faster simulation, and it is not part of the real FDDI protocol. The interconnected model is not allowed to use this shared flag, because of the tokens belonging to different LANs.

### 3. Results

Figure 20 shows the local throughput observed in the surface LAN. The random generation of destination addresses with uniform distribution led a considerable amount of traffic to the local stations. Figure 21 is the throughput directed to the command link from the surface LAN. In both cases, 90 percent of the traffic belongs to the asynchronous transmission as expected.

Figures 22 and 23 show the traffic contributed to the whole network by the collection platform LAN. While there is no local traffic appearing within this LAN, all traffic is directed to the return link with the specified proportion of asynchronous transmission.

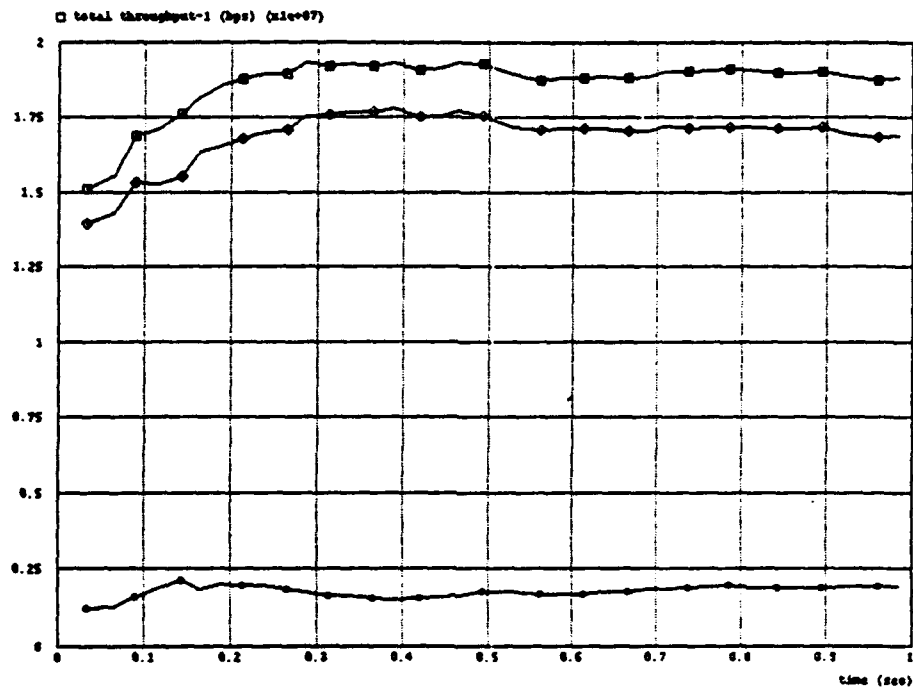


Figure 20: Local throughput of surface LAN.

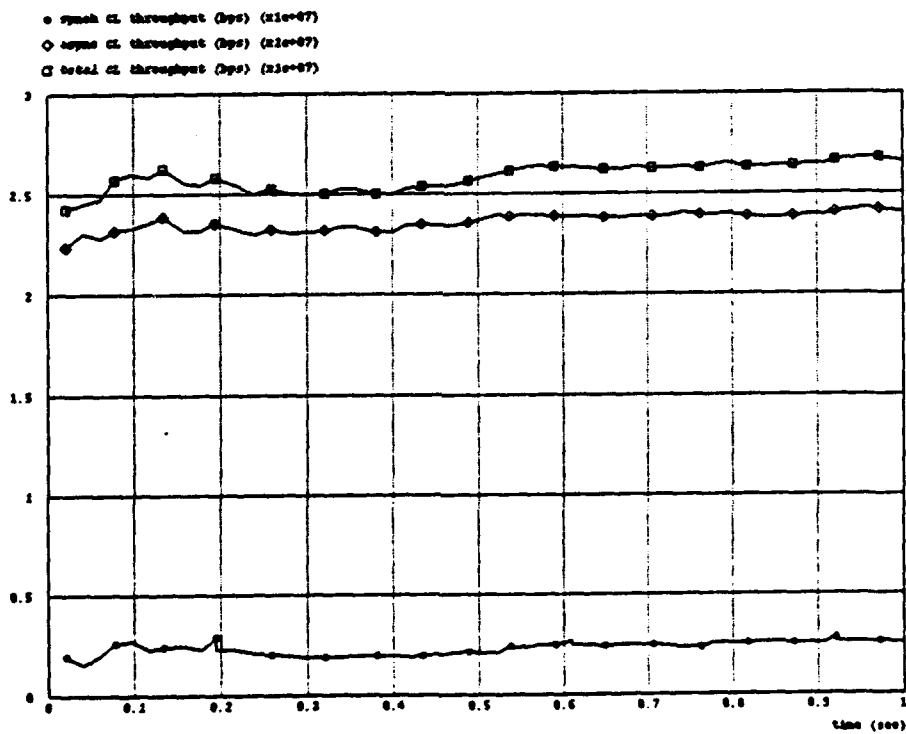


Figure 21: Traffic directed to the command link.

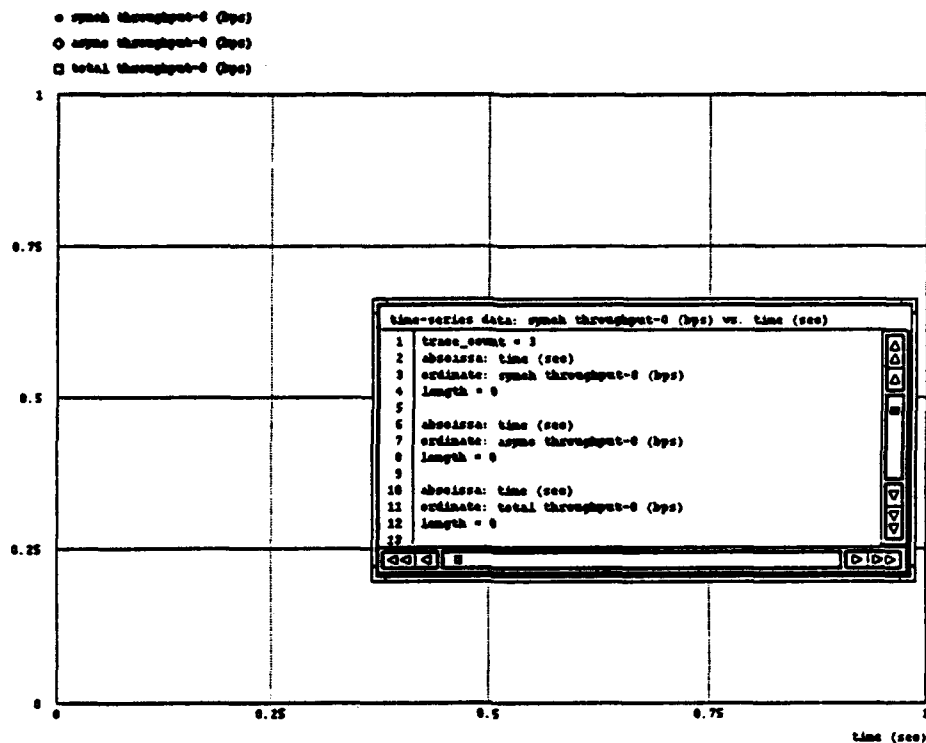


Figure 22: Local throughput of collection platform LAN.

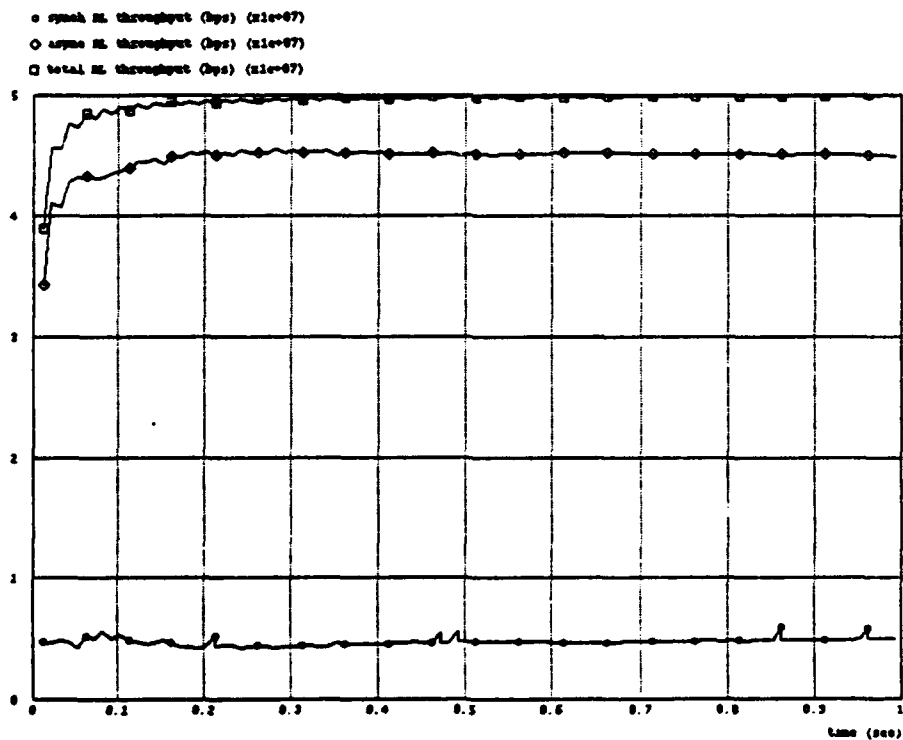


Figure 23: Throughput directed to the return link.

## **D. RETURN LINK PERFORMANCE**

### **1. Overview**

The station acting as the CPNI in the collection platform LAN is capable of buffering packets in its `llc_sink` node. The buffering is realized in terms of a transmission capacity allocation algorithm for the efficient utilization of the return link. The tests run in this section are related to the evaluation of these algorithms and the link throughput under jamming.

Two tests are described here. The first test is monitoring of packets accumulated in the buffers with two distinct allocation algorithms. The second test is observing the multiple link throughput at both ends of the CDL link.

### **2. First Test**

#### **a. Setup**

This test is based on the evaluation of different types of load balancing over the multiple links. Both FDDI LANs are configured with the same parameters as it is done for the traffic monitoring tests. In the simulation environment file, the "load balancing algorithm" attribute is also set appropriately for two different runs. This attribute is set to "0" for the circular allocation algorithm as it is set to "1" for the empty selection. Besides, the `llc_sink` subqueue capacities are set to hold a maximum of 200 packets. This was intended to observe the buffers' saturation and overflow as the traffic is directed to the return link at a constant rate.

#### **b. Results**

Figure 24 shows the accumulation of packets in the buffers when the circular allocation algorithm is in use. As seen, the first transmission channel having the lowest data rate causes its buffer to fill up quickly. Consequently, the

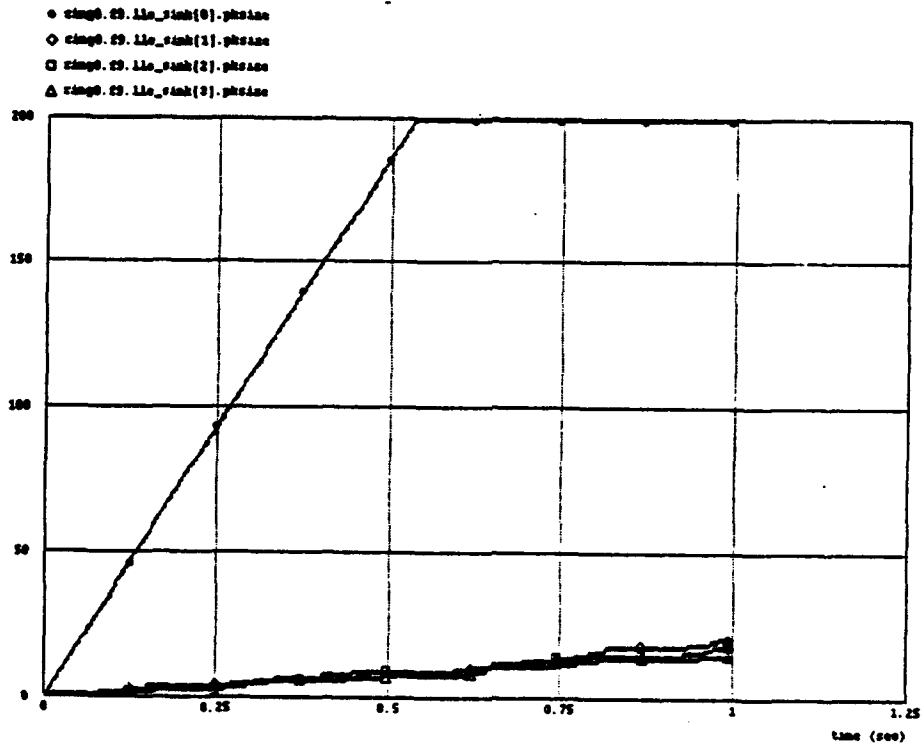


Figure 24: Accumulation of packets on the CPNI buffers with circular allocation.

continuous overflow of this buffer is unavoidable with the specified buffer size as depicted in Figure 25.

Conversely, the empty selection algorithm results in equal utilization of buffers for the same traffic load without introducing any overflow as shown in Figure 26.

The queuing delays of the packets observed for each buffer using both algorithms are shown in Figures 27 and 28. As compared to the empty selection algorithm, the circular load balancing provides a faster transmission rate for the channels having higher data rates and introduce a longer queuing delay for

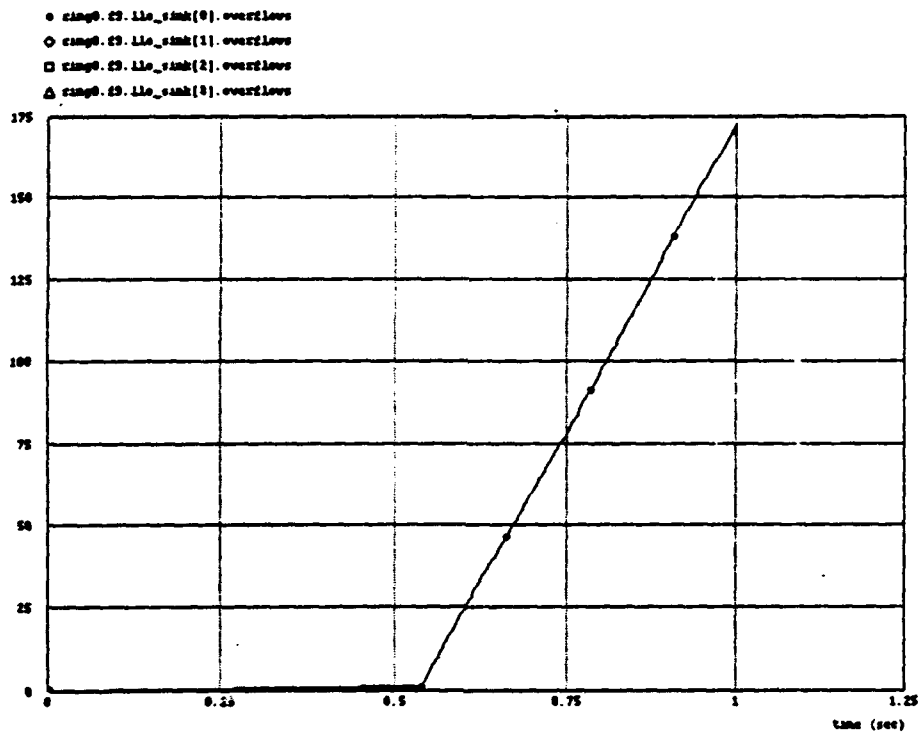


Figure 25: Buffer overflows in the CPNI with circular allocation.

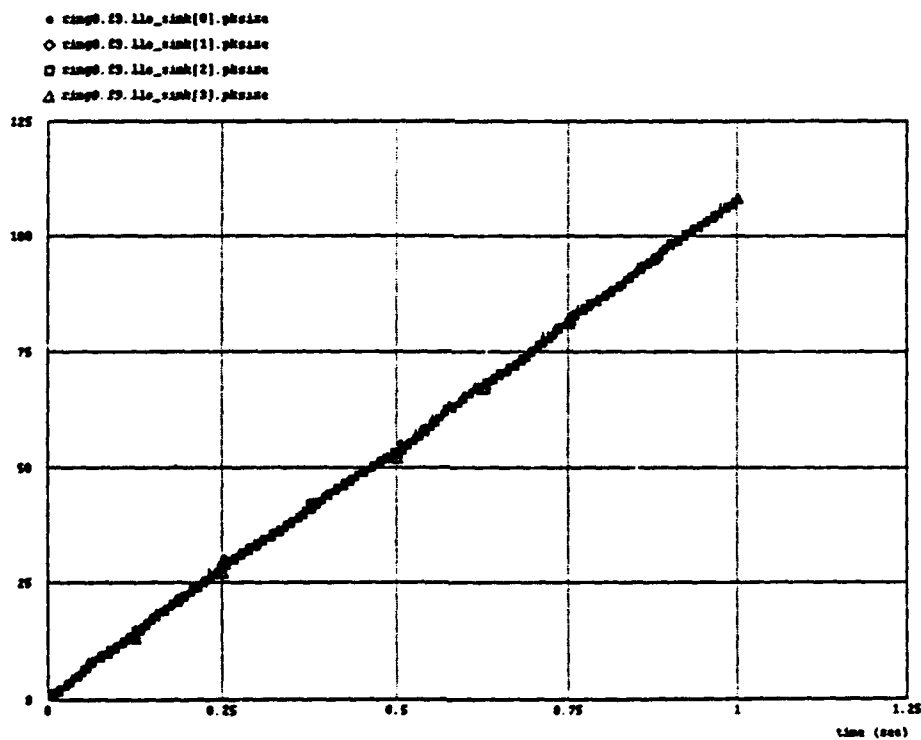


Figure 26: Accumulation of packets in the CPNI buffers with empty selection.



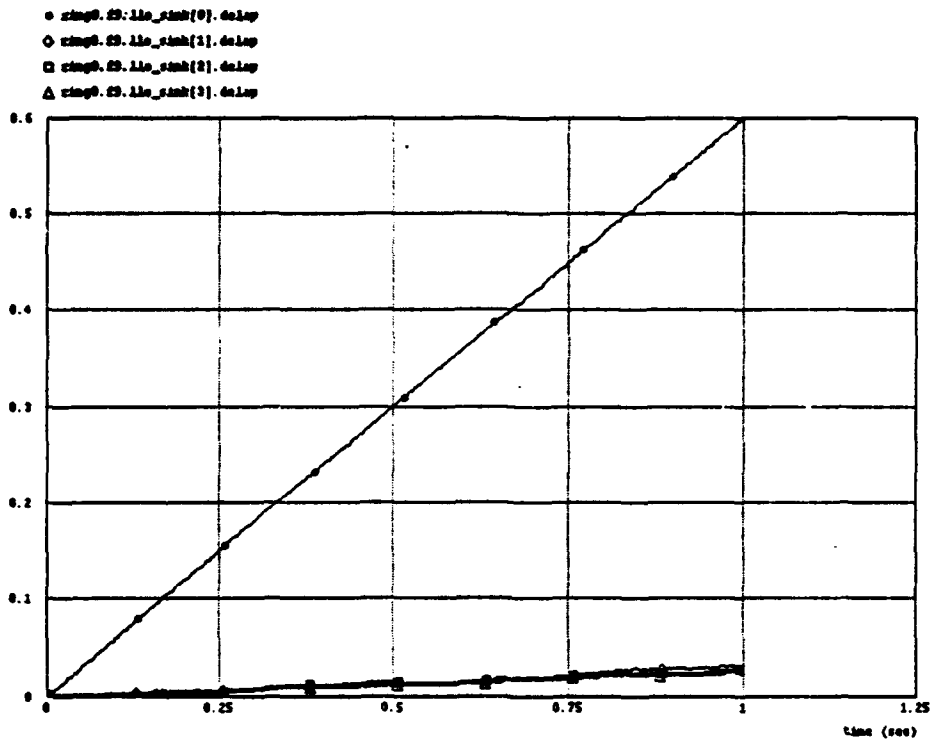


Figure 27: Queuing delay of the CPNI buffers with circular allocation.

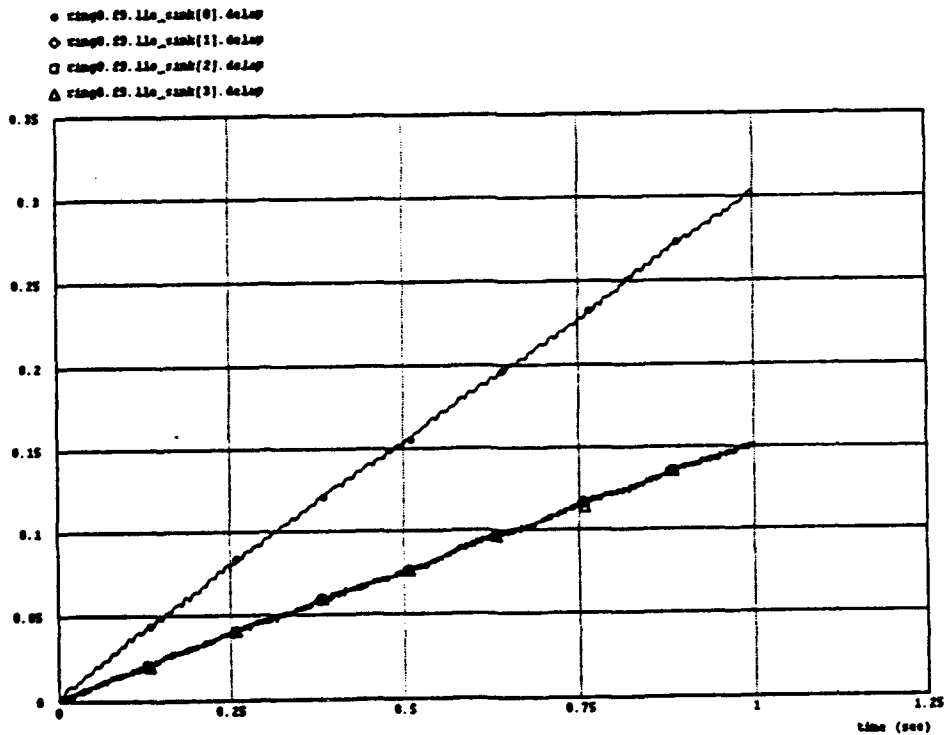


Figure 28: Queuing delay of the CPNI buffers with empty selection.

the slower channel. Primarily, the insufficient buffer sizes are the major drawback in the circular allocation algorithm, even though multiple links are still efficiently utilized.

As opposed to the circular allocation, empty selection algorithm does not require greater buffer sizes. It also offers an efficient utilization scheme with the penalty of more queuing delay for the channels with higher transmission capacities.

### **3. Second Test**

#### **a. Setup**

This test is intended to verify the effects of jamming models on the return link. The same simulation configuration is used in this test as before. Moreover, as defined in Chapter III, the jamming related attributes are set up properly to model two different jammers. The environment file used for the pulsed jammer simulation is provided in Appendix H. For the channel-swept jammer simulation, the same type of jamming attributes are set to identical values except "init\_jam\_offset". This attribute is doubled for each channel so that consecutive pulses can be created with proportional offsets. The jamming related attributes of the environment file for the channel-swept jammer simulation is in Appendix I.

The propagation delay of the CDL model is also specified as a typical value of 60 ms in the "delay" attribute of each link.

#### **b. Results**

Figure 29 shows the throughput of each transmitter channel in the CPNI end of the return link. As expected, identical number of bits are injected into the return link for the channels having the same data rates. The packets awaiting transmission in the buffers do not contribute to this statistic.

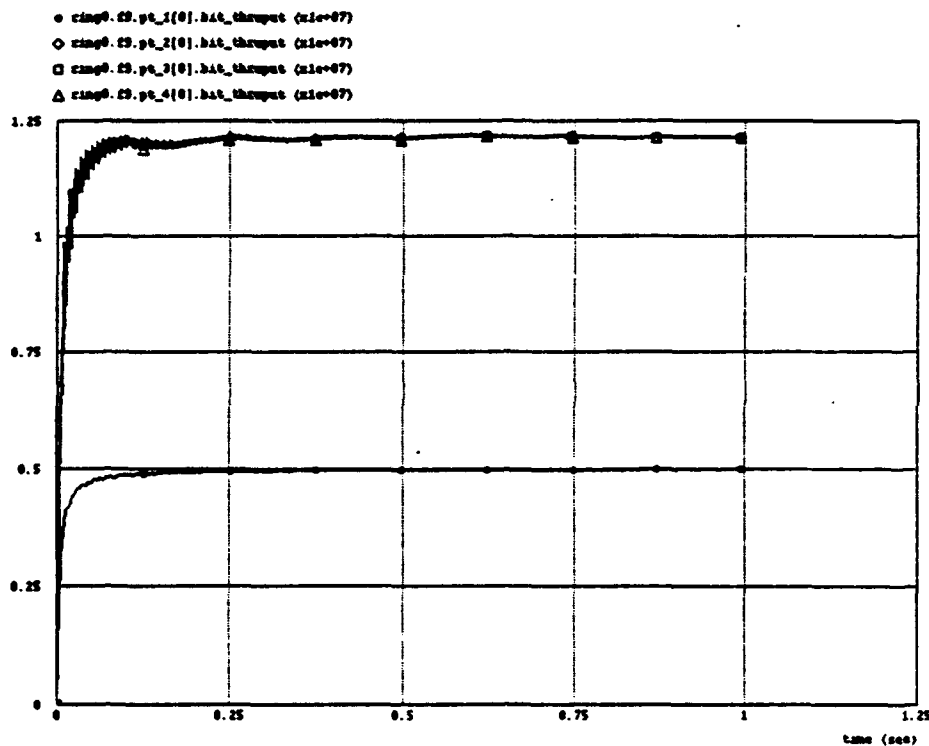


Figure 29: Throughput at the transmit-end of the return link.

The average BERs introduced by the pulsed jammer and the channel-swept jammer are depicted in Figures 30 and 31, respectively. As monitored in the receivers of the SPNI, all of the links are equally degraded in the channel-swept jammer's case due to the consecutive pulses. In the presence of pulsed jammer, the links exposed to the longer duration of jamming pulses are affected more severely.

Consequently, since the packets are corrupted due to jamming during the transmission, the throughput attained in the receive-end is not the same as the transmit-end of the return link. The effects of two different jamming models on the received throughput can be seen in Figures 32 and 33.

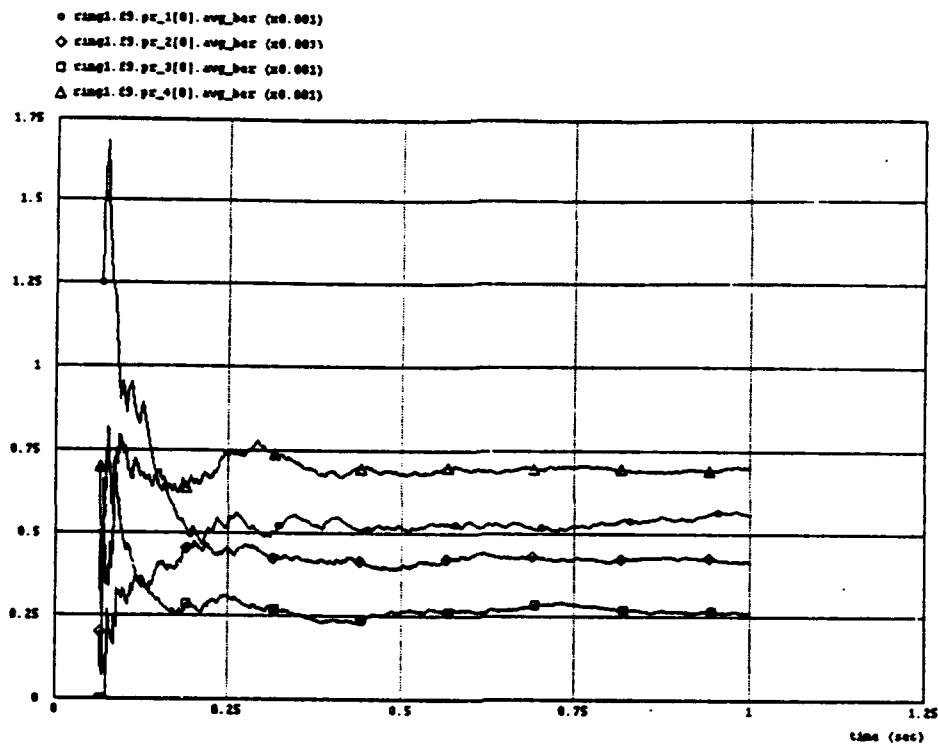


Figure 30: Average BER of the return link caused by pulsed jammer.

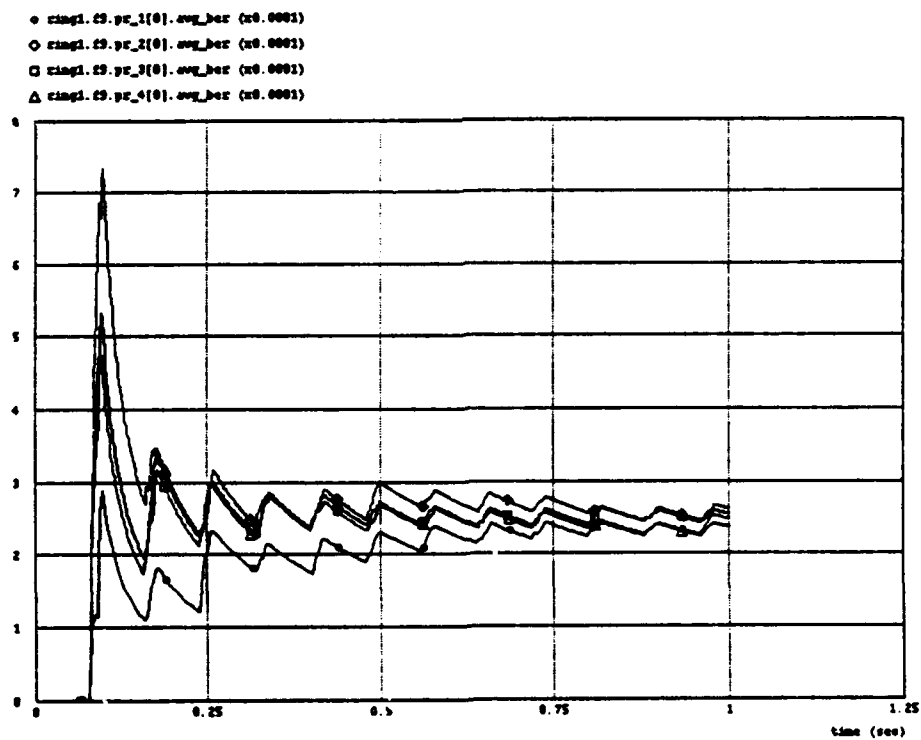


Figure 31: Average BER of the return link caused by channel-swept jammer.

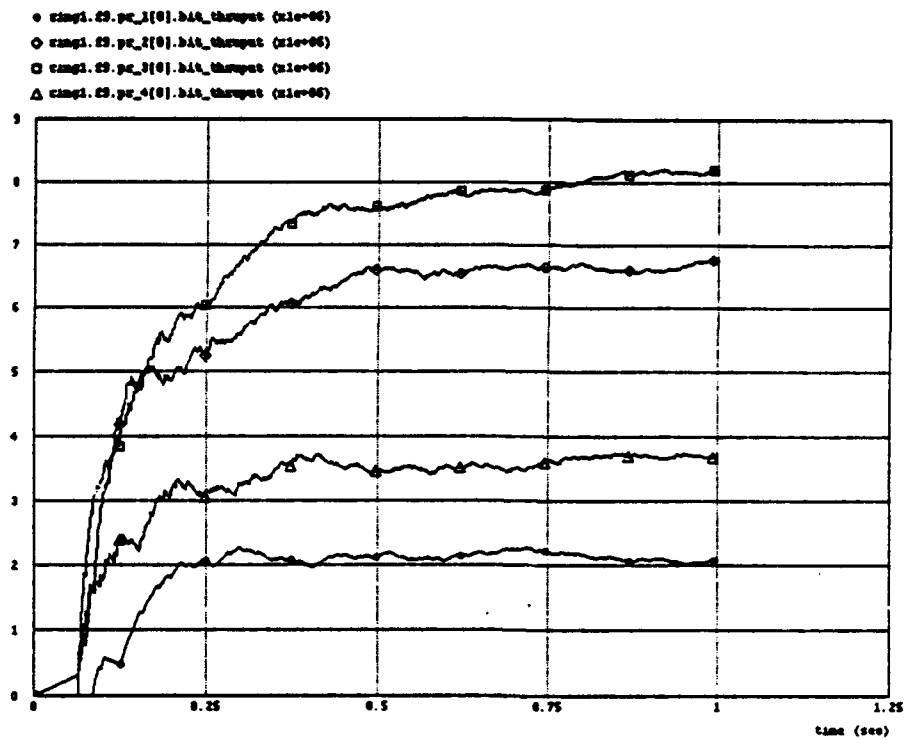


Figure 32: Effect of pulsed jammer on the return link throughput.

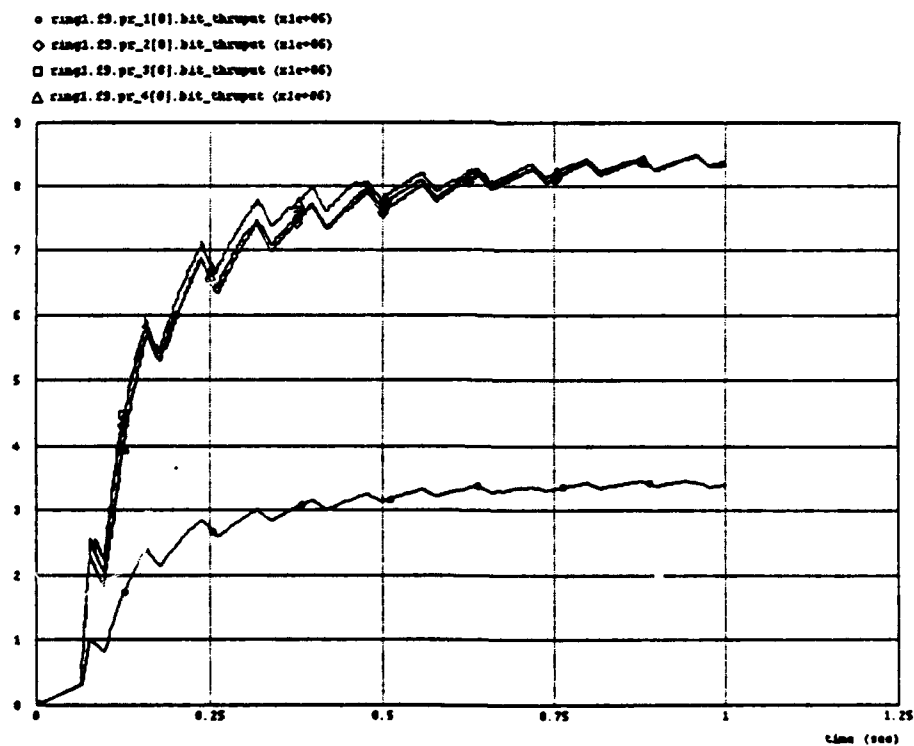


Figure 33: Effect of channel-swept jammer on the return link throughput.

## E. COMMAND LINK PERFORMANCE

Although the command link is modeled with respect to the CDL system requirements, its performance is not studied in as much detail as the return link in our simulations. As described before, the command link employs a relatively low fixed data rate in its channel hierarchy compared to the return link. Since multiple links are not modeled for this particular link, no algorithm for transmission allocation is developed in `llc_sink` node of the SPNI. As a result, the packets waiting for transmission to the collection platform LAN are accumulated in the single buffer of the SPNI.

Figure 34 shows the number of packets buffered over time. The accumulation is faster due to the lower transmission capacity of the command link. This is also emphasized with the buffer queuing delay as depicted in Figure 35. The asymmetric data rates existing in the CDL link require a greater buffer size for the SPNI. This buffer size must be evaluated with respect to the amount of traffic injected into the CDL by the surface LAN.

Similarly, the information transferred over the command link is monitored in both NIs of the interconnected model. Figure 36 is a simple illustration of the link throughput in transmit and receive-ends when the command link is not exposed to any jamming.

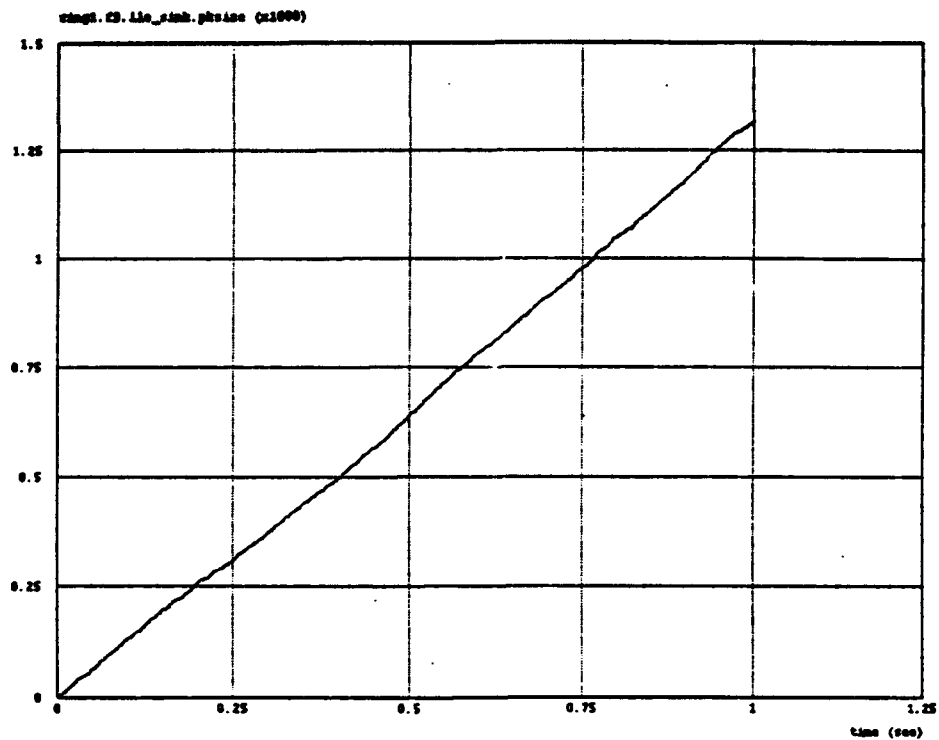


Figure 34: Accumulation of packets in the SPNI buffer.

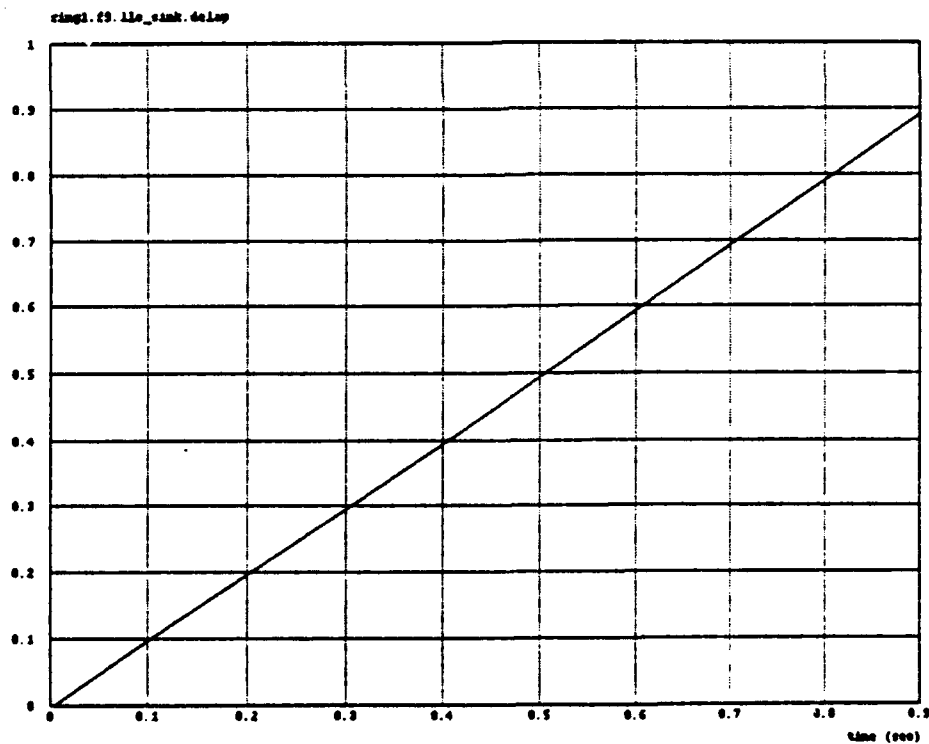


Figure 35: Queuing delay of the SPNI.

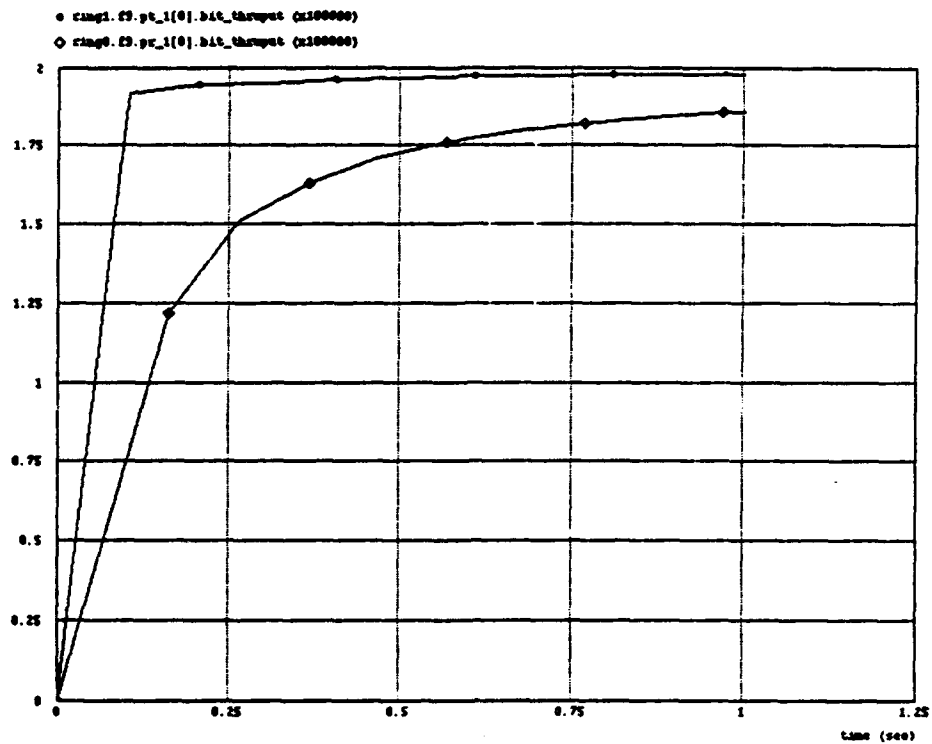


Figure 36: Throughput at the transmit and receive-ends of the command link.



## **V. CONCLUSIONS AND RECOMMENDATIONS**

### **A. CONCLUSIONS**

In this thesis, we have presented a performance analysis of a network interface device that interconnects two remote FDDI LANs using the CDL. The efficiency of the CDL model is evaluated with two different jamming patterns that faithfully represent the real environmental conditions. The distribution of load over the multiple links is also investigated using different transmission capacity allocation algorithms so that the overall link utilization can be maximized. The need for such algorithms is based on the near-capacity return link traffic load that is expected in the interconnected network. It is seen that the circular allocation algorithm is able to provide a faster transmission for the channels having higher data rates with the penalty of increased buffer size for the slower channels. When the empty selection algorithm is in use, identical buffer sizes can be selected for any transmission channel, if the drawback of longer queuing delays for faster channels are acceptable.

This thesis lays the foundation for simulating the multilink point-to-point protocol over the CDL in the further development of the network interface features for the CDL project.

### **B. RECOMMENDATIONS**

Further developments of our model will include:

1. Simulation of a link monitoring protocol over the return and command link. This capability will facilitate feedback to the LAN applications

(end-systems) about the link status and permit end-to-end performance analysis.

2. Incorporation of a forwarding/filtering database in the NI.
3. Incorporation of a traffic generation in the LAN end-systems that is closer to the characteristics of real traffic patterns.
4. Accurate estimation of NI buffer capacity for individual channels with various effective BERs over the link.

Since the tool-specific token acceleration feature is disabled during our simulations, a new mechanism need to be employed for a faster simulation in a larger network model.

# APPENDIX A

## CPNI SOURCE "C" CODE

"cp\_fddi\_gen.pr.c"

The line numbering in this appendix is within this thesis only, and does not correspond with that seen in OPNET's text editors.

```
1  /* Process model C form file: cp_fddi_gen.pr.c */
2  /* Portions of this file Copyright (C) MIL 3, Inc. 1992 */

3  /* OPNET system definitions */
4  #include <opnet.h>
5  #include "cp_fddi_gen.pr.h"
6  FSM_EXT_DECS

7  /* Header block */
8  #define MAC_LAYER_OUT_STREAM      0
9  #define LLC_SINK_OUT_STREAM      1 /*18APR94*/

10 /* define possible service classes for frames */
11 #define FDDI_SVC_ASYNC            0
12 #define FDDI_SVC_SYNC            1

13 /* define token classes */
14 #define FDDI_TK_NONRESTRICTED     0
15 #define FDDI_TK_RESTRICTED       1

16 /* State variable definitions */
17 typedef struct
18 {
19     FSM_SYS_STATE
20     Distribution*          sv_inter_dist_ptr;
21     Distribution*          sv_len_dist_ptr;
22     Distribution*          sv_dest_dist_ptr;
23     Distribution*          sv_pkt_priority_ptr;
24     Objid                  sv_mac_objid;
25     Objid                  sv_my_id;
26     int                    sv_low_dest_addr;
27     int                    sv_high_dest_addr;
```

```

28     int                sv_station_addr;
29     int                sv_src_addr;
30     int                sv_low_pkt_priority;
31     int                sv_high_pkt_priority;
32     double             sv_arrival_rate;
33     double             sv_mean_pk_len;
34     double             sv_async_mix;
35     Ici*               sv_mac_iciptr;
36     Ici*               sv_mac_iciptri;
37     Ici*               sv_llc_ici_ptr;
38     Packet*            sv_pkptri;
39     } cp_fddi_gen_state;

40 #define pr_state_ptr      ((cp_fddi_gen_state*) SimI_Mod_State_Ptr)
41 #define inter_dist_ptr    pr_state_ptr->sv_inter_dist_ptr
42 #define len_dist_ptr      pr_state_ptr->sv_len_dist_ptr
43 #define dest_dist_ptr     pr_state_ptr->sv_dest_dist_ptr
44 #define pkt_priority_ptr  pr_state_ptr->sv_pkt_priority_ptr
45 #define mac_objid         pr_state_ptr->sv_mac_objid
46 #define my_id             pr_state_ptr->sv_my_id
47 #define low_dest_addr     pr_state_ptr->sv_low_dest_addr
48 #define high_dest_addr    pr_state_ptr->sv_high_dest_addr
49 #define station_addr      pr_state_ptr->sv_station_addr
50 #define src_addr          pr_state_ptr->sv_src_addr
51 #define low_pkt_priority  pr_state_ptr->sv_low_pkt_priority
52 #define high_pkt_priority pr_state_ptr->sv_high_pkt_priority
53 #define arrival_rate      pr_state_ptr->sv_arrival_rate
54 #define mean_pk_len       pr_state_ptr->sv_mean_pk_len
55 #define mac_iciptr        pr_state_ptr->sv_mac_iciptr
56 #define mac_iciptri       pr_state_ptr->sv_mac_iciptri
57 #define llc_ici_ptr       pr_state_ptr->sv_llc_ici_ptr
58 #define pkptri            pr_state_ptr->sv_pkptri

59 /* Process model interrupt handling procedure */

60 void
61 cp_fddi_gen ()
62 {
63     Packet *pkptri;
64     int     pklen;
65     int     dest_addr;
66     int     i, restricted;
67     int     pkt_prio;

68     FSM_ENTER (cp_fddi_gen)

```

```

69     FSM_BLOCK_SWITCH
70     {
71         /*-----*/
72         /** state (INIT) enter executives **/
73         FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "INIT")
74         {
75             /* determine id of own processor to use in finding attrs */
76             my_id = op_id_self ();

77             /* determine address range for uniform desination assignment */
78             op_ima_obj_attr_get (my_id, "low dest address", &low_dest_addr);
79             op_ima_obj_attr_get (my_id, "high dest address", &high_dest_addr);

80             /* determine object id of connected 'mac' layer process */
81             mac_objid = op_topo_assoc (my_id, OPC_TOPO_ASSOC_OUT,
82                                     OPC_OBJMTYPE_MODULE,
83                                     MAC_LAYER_OUT_STREAM);

84             /* determine the address assigned to it */
85             /* which is also the address of this station */
86             op_ima_obj_attr_get (mac_objid, "station_address", &station_addr);

87             /* set up a distribution for generation of addresses */
88             dest_dist_ptr = op_dist_load ("uniform_int", low_dest_addr,
89             high_dest_addr);

90             /* added 26DEC93 */
91             /* determine priority range for uniform traffic generation */
92             op_ima_obj_attr_get (my_id, "high pkt priority", &high_pkt_priority);
93             op_ima_obj_attr_get (my_id, "low pkt priority", &low_pkt_priority);

94             /* set up a distribution for generation of priorities */
95             pkt_priority_ptr = op_dist_load ("uniform_int", low_pkt_priority,
96             high_pkt_priority);

97             /* above added 26DEC93 */

98             /* also determine the arrival rate for packet generation */
99             op_ima_obj_attr_get (my_id, "arrival rate", &arrival_rate);

100            /* determine the mix of asynchronous and synchronous */
101            /* traffic. This is expressed as the proportion of */
102            /* asynchronous traffic. i.e a value of 1.0 indicates */
103            /* that all the produced traffic shall be asynchronous. */
104            op_ima_obj_attr_get (my_id, "async_mix", &async_mix);

105            /* set up a distribution for arrival generations */

```

```

106         if (arrival_rate != 0.0)
107         {
108             /* arrivals are exponentially distributed, with given mean */
109             inter_dist_ptr = op_dist_load ("constant", 1.0 / arrival_rate,
110 0.0);

111             /* determine the distribution for packet size */
112             op_ima_obj_attr_get (my_id, "mean pk length",
113 &mean_pk_len);

114             /* set up corresponding distribution */
115             len_dist_ptr = op_dist_load ("constant", mean_pk_len, 0.0);

116             /* designate the time of first arrival */
117             fddi_gen_schedule ();

118             /* set up an interface control information (ICI) structure */
119             /* to communicate parameters to the mac layer process */
120             /* (it is more efficient to set one up now and keep it */
121             /* as a state variable than to allocate one on each packet
122 xfer) */
123             mac_iciptr = op_ici_create ("fddi_mac_req");
124         }

125     }

126     /** blocking after enter executives of unforced state. **/
127     FSM_EXIT (1, cp_fddi_gen)

128     /** state (INIT) exit executives **/
129     FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "INIT")
130     {
131     }

132     /** state (INIT) transition processing **/
133     FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
134     /*-----*/

135     /** state (ARRIVAL) enter executives **/
136     FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "ARRIVAL")
137     {
138         /* This station should receive frames from the other lan as long as */
139         /* there are frames in the input streams addressed to this lan */
140         /* check if the interrupt type is stream interrupt */
141         if (op_intrpt_type() == OPC_INTRPT_STRM)
142         {

```

```

143      /* if it is, get the packet in the input stream causing interrupt */
144      pkptr1 = op_pk_get(op_intrpt_strm());
145      /* get the destination address of the frame */
146      /* 16APR94 */
147      op_pk_nfd_get(pkptr1, "dest_addr", &dest_addr);
148      /* check if this frame destined for the local bridge station */
149      if(dest_addr == station_addr)
150          /* if it is, send the packet to llc_sink directly */
151          /* in order to prevent overhead of mac access */
152          op_pk_send(pkptr1,
153      LLC_SINK_OUT_STREAM); /*19APR94*/
154      else
155          /* this packet is to send to mac */
156          {
157              /* determine the source address of the frame */
158              op_pk_nfd_get(pkptr1, "src_addr", &src_addr);
159              /* set up an ICI structure to communicate parameters to */
160              /* MAC layer process */
161              mac_iciptr1 = op_ici_create("fddi_mac_req");
162              /* place the original source address into the ICI */
163              /* 16APR94 */
164              /* "fddi_mac_req" is modified so that it contains the original
165              */
166              /* source address from the remote lan */
167              op_ici_attr_set(mac_iciptr1, "src_addr", src_addr);
168              /* place the destination address into the ICI */
169              /*12APR94*/
170              op_ici_attr_set(mac_iciptr1, "dest_addr", dest_addr);
171              /* assign the service class and requested token class */
172              /* At this moment the frames coming from the remote lan
173              are assumed to have*/
174              /* the same priority as synchronous frames in order not to
175              accumulate */
176              /* packets on the bridge station mac and instead to deliver
177              their destinations */
178              /* as soon as possible */
179              op_pk_nfd_set(pkptr1, "pri", 8);
180              op_ici_attr_set(mac_iciptr1, "svc_class",
181      FDDI_SVC_SYNC);
182              op_ici_attr_set(mac_iciptr1, "pri", 8);
183              op_ici_attr_set(mac_iciptr1, "tk_class",
184      FDDI_TK_NONRESTRICTED);
185              /* send the packet coupled with the ICI */
186              op_ici_install(mac_iciptr1);
187              op_pk_send(pkptr1, MAC_LAYER_OUT_STREAM);
188          }
189      }
190      /* otherwise, generate the frame :12APR94 */
191      else
192      {
193          /* determine the length of the packet to be generated */

```

```

194         pklen = op_dist_outcome (len_dist_ptr);

195         /* determine the destination */
196         /* dont allow this station's address as a possible outcome */
197         gen_packet:
198         dest_addr = op_dist_outcome (dest_dist_ptr);
199         if (dest_addr != -1 && dest_addr == station_addr)
200             goto gen_packet;

201         /* 26DEC94 & 29JAN94: determine its priority */
202         pkt_prio = op_dist_outcome (pkt_priority_ptr);

203         /* create a packet to send to mac */
204         pkptr = op_pk_create_fmt ("fddi_llc_fr");

205         /* assign its overall size. */
206         op_pk_total_size_set (pkptr, pklen);

207         /* assign the time of creation */
208         op_pk_nfd_set (pkptr, "cr_time", op_sim_time ());

209         /* place the destination address into the ICI */
210         /* (the protocol_type field will default) */
211         op_ici_attr_set (mac_iciptr, "dest_addr", dest_addr);

212         /* place the source address into the ICI */ /* 17APR94 */
213         op_ici_attr_set (mac_iciptr, "src_addr", station_addr);

214         /* assign the priority, and requested token class */
215         /* also assign the service class; 29JAN94: the fddi_llc_fr */
216         /* format is modified to include a "pri" field. */
217         if (op_dist_uniform (1.0) <= async_mix)
218             {
219                 op_pk_nfd_set (pkptr, "pri", pkt_prio); /* 29JAN94 */
220                 op_ici_attr_set (mac_iciptr, "svc_class",
221 FDDI_SVC_ASYNC);
222                 op_ici_attr_set (mac_iciptr, "pri", pkt_prio); /* 29JAN94 */
223             }
224         else{
225             op_pk_nfd_set (pkptr, "pri", 8); /* 29JAN94 */
226             op_ici_attr_set (mac_iciptr, "svc_class",
227 FDDI_SVC_SYNC);
228             op_ici_attr_set (mac_iciptr, "pri", 8); /* 29JAN94 */
229         }

230         /* Request only nonrestricted tokens after transmission */
231         op_ici_attr_set (mac_iciptr, "tk_class",
232 FDDI_TK_NONRESTRICTED);

233         /* Having determined priority, assign it; 26DEC93 */
234         /* op_ici_attr_set (mac_iciptr, "pri", pkt_prio); */

```



```

235          /* send the packet coupled with the ICI */
236          op_ici_install (mac_iciptr);
237          /* check if destination address is in the remote lan */
238          if(dest_addr > 9)
239              /* if it is, this packet is to send llc_sink directly */
240              op_pk_send (pkptr, LLC_SINK_OUT_STREAM);
241          /*18APR94*/
242          else
243              /* if not, the packet is destined for local lan, so send to mac
244          */
245              op_pk_send (pkptr, MAC_LAYER_OUT_STREAM);

246          /* schedule the next arrival */
247          fddi_gen_schedule ();
248      }
249  }

250      /** blocking after enter executives of unforced state. **/
251      FSM_EXIT (3,cp_fddi_gen)

252      /** state (ARRIVAL) exit executives **/
253      FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "ARRIVAL")
254      {
255      }

256      /** state (ARRIVAL) transition processing **/
257      FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
258      /*-----*/

259  }

260      FSM_EXIT (0,cp_fddi_gen)
261  }

262  void
263  cp_fddi_gen_svar (prs_ptr,var_name,var_p_ptr)
264      cp_fddi_gen_state      *prs_ptr;
265      char                    *var_name, **var_p_ptr;
266  {

267      FIN (cp_fddi_gen_svar (prs_ptr))

```

```

268     *var_p_ptr = VOS_NIL;
269     if (Vos_String_Equal ("inter_dist_ptr" , var_name))
270         *var_p_ptr = (char *) (&prs_ptr->sv_inter_dist_ptr);
271     if (Vos_String_Equal ("len_dist_ptr" , var_name))
272         *var_p_ptr = (char *) (&prs_ptr->sv_len_dist_ptr);
273     if (Vos_String_Equal ("dest_dist_ptr" , var_name))
274         *var_p_ptr = (char *) (&prs_ptr->sv_dest_dist_ptr);
275     if (Vos_String_Equal ("pkt_priority_ptr" , var_name))
276         *var_p_ptr = (char *) (&prs_ptr->sv_pkt_priority_ptr);
277     if (Vos_String_Equal ("mac_objid" , var_name))
278         *var_p_ptr = (char *) (&prs_ptr->sv_mac_objid);
279     if (Vos_String_Equal ("my_id" , var_name))
280         *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
281     if (Vos_String_Equal ("low_dest_addr" , var_name))
282         *var_p_ptr = (char *) (&prs_ptr->sv_low_dest_addr);
283     if (Vos_String_Equal ("high_dest_addr" , var_name))
284         *var_p_ptr = (char *) (&prs_ptr->sv_high_dest_addr);
285     if (Vos_String_Equal ("station_addr" , var_name))
286         *var_p_ptr = (char *) (&prs_ptr->sv_station_addr);
287     if (Vos_String_Equal ("src_addr" , var_name))
288         *var_p_ptr = (char *) (&prs_ptr->sv_src_addr);
289     if (Vos_String_Equal ("low_pkt_priority" , var_name))
290         *var_p_ptr = (char *) (&prs_ptr->sv_low_pkt_priority);
291     if (Vos_String_Equal ("high_pkt_priority" , var_name))
292         *var_p_ptr = (char *) (&prs_ptr->sv_high_pkt_priority);
293     if (Vos_String_Equal ("arrival_rate" , var_name))
294         *var_p_ptr = (char *) (&prs_ptr->sv_arrival_rate);
295     if (Vos_String_Equal ("mean_pk_len" , var_name))
296         *var_p_ptr = (char *) (&prs_ptr->sv_mean_pk_len);
297     if (Vos_String_Equal ("async_mix" , var_name))
298         *var_p_ptr = (char *) (&prs_ptr->sv_async_mix);
299     if (Vos_String_Equal ("mac_iciptr" , var_name))
300         *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr);
301     if (Vos_String_Equal ("mac_iciptr1" , var_name))
302         *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr1);
303     if (Vos_String_Equal ("llc_ici_ptr" , var_name))
304         *var_p_ptr = (char *) (&prs_ptr->sv_llc_ici_ptr);
305     if (Vos_String_Equal ("pkpptr1" , var_name))
306         *var_p_ptr = (char *) (&prs_ptr->sv_pkpptr1);

307     FOUT;
308 }

```

```

309 void
310 cp_fddi_gen_diag ()
311 {
312     Packet      *pkpptr;

```

```

313     int          pklen;
314     int          dest_addr;
315     int          i, restricted;
316     int          pkt_prio;

317     FIN (cp_fddi_gen_diag ())

318     FOUT;
319 }

320 void
321 cp_fddi_gen_terminate ()
322 {
323     Packet        *pkptr;
324     int          pklen;
325     int          dest_addr;
326     int          i, restricted;
327     int          pkt_prio;

328     FIN (cp_fddi_gen_terminate ())

329     FOUT;
330 }

331 Compcode
332 cp_fddi_gen_init (pr_state_pptr)
333     cp_fddi_gen_state      **pr_state_pptr;
334 {
335     static VosT_Cm_Obtype    obtype = OPC_NIL;

336     FIN (cp_fddi_gen_init (pr_state_pptr))

337     if (obtype == OPC_NIL)
338     {
339         if (Vos_Catmem_Register ("proc state vars (cp_fddi_gen)",
340             sizeof (cp_fddi_gen_state), Vos_Nop, &obtype) == VOSC_FAILURE)
341             FRET (OPC_COMPCODE_FAILURE)
342     }

343     if ((*pr_state_pptr = (cp_fddi_gen_state*) Vos_Catmem_Alloc (obtype, 1)) ==
344     OPC_NIL)
345         FRET (OPC_COMPCODE_FAILURE)
346     else
347         {

```

```

348      (*pr_state_pptr)->current_block = 0;
349      FRET (OPC_COMPCODE_SUCCESS)
350  }
351  }

352  /* static added 2DEC93, on advice from MIL3 */
353  static
354  fddi_gen_schedule ()
355  {
356      double          inter_time;

357      /* obtain an interarrival period according to the */
358      /* prescribed distribution */
359      inter_time = op_dist_outcome (inter_dist_ptr);

360      /* schedule the arrival of next generated packet */
361      op_intrpt_schedule_self (op_sim_time () + inter_time, 0);
362  }

```

## APPENDIX B

### CPNI MAC "C" CODE

"cp\_fddi\_mac.pr.c"

The line numbering in this appendix is within this thesis only, and does not correspond with that seen in OPNET's text editors.

```
1  /* Process model C form file: cp_fddi_mac.pr.c */
2  /* Portions of this file Copyright (C) MIL 3, Inc. 1992 */

3  /* OPNET system definitions */
4  #include <opnet.h>
5  #include "cp_fddi_mac.pr.h"
6  FSM_EXT_DECS

7  /* Header block */
8  /* Define a timer structure used to implement */
9  /* the TBT and THT timers. The primitives defined to */
10 /* operate on these timers can be found in the */
11 /* function block of this process model. */
12 typedef struct
13 {
14     int enabled;
15     double start_time;
16     double accum;
17     double target_accum;
18 } FddiT_Timer;

19 /* Declare certain primitives dealing with timer.s */
20 double fddi_timer_remaining ();
21 FddiT_Timer* fddi_timer_create ();
22 double fddi_timer_value ();

23 /* Scratch strings for trace statements */
24 char str0 [512], str1 [512];

25 /* define constants particular to this implementation */
26 #define FDDI_MAX_STATIONS 512

27 /* define possible values for the frame control field */
28 #define FDDI_FC_FRAME 0
```

```

29 #define FDDI_FC_TOKEN 1

30 /* define possible service classes for frames */
31 #define FDDI_SVC_ASYNC 0
32 #define FDDI_SVC_SYNC 1

33 /* define input stream indices */
34 #define FDDI_LLC_STM_IN 1
35 #define FDDI_PHY_STM_IN 0

36 /* define output stream indices */
37 #define FDDI_LLC_STM_OUT 1
38 #define FDDI_PHY_STM_OUT 0

39 /* define token classes */
40 #define FDDI_TK_NONRESTRICTED 0
41 #define FDDI_TK_RESTRICTED 1

42 /* Ring Constants */
43 #define FDDI_TX_RATE 1.0e+08
44 #define FDDI_SA_SCAN_TIME 28.0e-08

45 /* Token transmission time: based on 6 symbols plus 16 symbols of preamble */
46 #define FDDIC_TOKEN_TX_TIME 88.0e-08

47 /* Codes used to differentiate remote interrupts */
48 #define FDDIC_TRT_EXPIRE 0
49 #define FDDIC_TK_INJECT 1

50 /* Define symbolic expressions used on transition */
51 /* conditions and in executive statements. */
52 #define TRT_EXPIRE \
53 (op_intrpt_type () == OPC_INTRPT_REMOTE && op_intrpt_code () == FDDIC_TRT_EXPIRE)

54 #define TK_RECEIVED \
55 phy_arrival && \
56 frame_control == FDDI_FC_TOKEN

57 #define RC_FRAME \
58 phy_arrival && \
59 frame_control == FDDI_FC_FRAME

60 #define FRAME_ARRIVAL \
61 op_intrpt_type () == OPC_INTRPT_STM && \
62 op_intrpt_stm () == FDDI_LLC_STM_IN

63 #define STRIP my_address == src_addr

64 /* Define the maximum value for ring_id. This is the */

```

```

65 /* maximum number of FDDI rings that can exist in a */
66 /* simulation. Note that if this number is changed, */
67 /* the initialization for fddi_claim_start below must */
68 /* also be modified accordingly. */
69 #define FDDI_MAX_RING_ID 8

70 /* Declare the operative TTRT value 'T_Opr' which is the final */
71 /* negotiated value of TTRT. This value is shared by all stations */
72 /* on a ring so that all agree on its value. */
73 double fddi_t_opr [FDDI_MAX_RING_ID];
74 #define Fddi_T_Opr (fddi_t_opr [ring_id])

75 /* This flag indicates that the negotiation for the final TTRT */
76 /* has not yet begun. It is statically initialized here, and */
77 /* is reset by the first station which modifies T_Opr. */
78 /* Initialize to 1 for all rings.*/
79 static
80 int fddi_claim_start [FDDI_MAX_RING_ID] = {1,1,1,1,1,1,1,1};
81 #define Fddi_Claim_Start (fddi_claim_start [ring_id])

82 /* Declare station latency parameters. */
83 /* These are true globals, so they do not need to be arrays. */
84 double Fddi_St_Latency;
85 double Fddi_Prop_Delay;

86 /* Declare globals for Token Acceleration Mechanism. */
87 /* Hop delay and token acceleration are true globals. */
88 double Fddi_Tk_Hop_Delay;
89 static
90 int Fddi_Tk_Accelerate = 1;

91 /* These are actually values shared by all nodes on a ring, */
92 /* so they must be defined as arrays. */
93 double fddi_tk_block_base_time [FDDI_MAX_RING_ID];
94 #define Fddi_Tk_Block_Base_Time (fddi_tk_block_base_time [ring_id])

95 int fddi_tk_block_base_station [FDDI_MAX_RING_ID];
96 #define Fddi_Tk_Block_Base_Station (fddi_tk_block_base_station [ring_id])

97 int fddi_tk_blocked [FDDI_MAX_RING_ID];
98 #define Fddi_Tk_Blocked (fddi_tk_blocked [ring_id])

99 int fddi_num_stations [FDDI_MAX_RING_ID];
100 #define Fddi_Num_Stations (fddi_num_stations [ring_id])

101 int fddi_num_registered [FDDI_MAX_RING_ID];
102 #define Fddi_Num_Registered (fddi_num_registered [ring_id])

103 Objid fddi_address_table [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
104 #define Fddi_Address_Table (fddi_address_table [ring_id])

```

```

105 /* Below is part of the OPBUG 2081 patch; FB ended here, before. -Nix */

106 /* Event handles for the TRT are maintained at a global level to */
107 /* allow token acceleration mechanism to adjust these as necessary */
108 /* when blocking and reinjecting the token. TRT_handle simply */
109 /* represents the TRT for the local MAC */
110 Evhandle fddi_trt_handle [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
111 #define Fddi_Trt_Handle (fddi_trt_handle [ring_id])
112 #define TRT_handle Fddi_Trt_Handle [my_address]

113 /* Similarly, the TRT data structure is maintained on a global level. */
114 FddiT_Timer* fddi_trt [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
115 #define Fddi_Trt (fddi_trt [ring_id])
116 #define TRT Fddi_Trt [my_address]

117 /* Registers to record the expiration time of each TRT when token is blocked. */
118 double fddi_trt_exp_time [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
119 #define Fddi_Trt_Exp_Time (fddi_trt_exp_time [ring_id])

120 /* the 'Late_Ct' flag is declared on a global level so that it can be */
121 /* set at the time where the token is injected back into the ring. */
122 int fddi_late_ct [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
123 #define Fddi_Late_Ct (fddi_late_ct [ring_id])
124 #define Late_Ct Fddi_Late_Ct [my_address]

125 /* Convenient macro for setting TRT for a given station and absolute time. */
126 #define TRT_SET(station_id,abs_time) \
127 fddi_timer_set (Fddi_Trt [station_id], abs_time - op_sim_time()); \
128 Fddi_Trt_Handle [station_id] = op_intrpt_schedule_remote (abs_time, \
129 FDDIC_TRT_EXPIRE, Fddi_Address_Table [station_id]);

130 /* State variable definitions */
131 typedef struct
132 {
133     FSM_SYS_STATE
134     int sv_ring_id;
135     FddiT_Timer* sv_THT;
136     double sv_T_Req;
137     double sv_T_Pri [8];
138     Objid sv_my_objid;
139     int sv_spawn_token;
140     int sv_my_address;
141     int sv_orig_src_addr;

```



```

141 Packet*          sv_tk_pkptr;
142 double           sv_sync_bandwidth;
143 double           sv_sync_pc;
144 int              sv_restricted;
145 int              sv_res_peer;
146 int              sv_tk_registered;
147 Ici*            sv_to_llc_ici_ptr;
148 int              sv_tk_trace_on;
149 } cp_fddi_mac_state;

150 #define pr_state_ptr      ((cp_fddi_mac_state*) SimI_Mod_State_Ptr)
151 #define ring_id           pr_state_ptr->sv_ring_id
152 #define THT              pr_state_ptr->sv_THT
153 #define T_Req            pr_state_ptr->sv_T_Req
154 #define T_Pri            pr_state_ptr->sv_T_Pri
155 #define my_objid         pr_state_ptr->sv_my_objid
156 #define spawn_token      pr_state_ptr->sv_spawn_token
157 #define my_address       pr_state_ptr->sv_my_address
158 #define orig_src_addr    pr_state_ptr->sv_orig_src_addr
159 #define tk_pkptr         pr_state_ptr->sv_tk_pkptr
160 #define sync_bandwidth   pr_state_ptr->sv_sync_bandwidth
161 #define sync_pc          pr_state_ptr->sv_sync_pc
162 #define restricted       pr_state_ptr->sv_restricted
163 #define res_peer         pr_state_ptr->sv_res_peer
164 #define tk_registered    pr_state_ptr->sv_tk_registered
165 #define to_llc_ici_ptr   pr_state_ptr->sv_to_llc_ici_ptr
166 #define tk_trace_on      pr_state_ptr->sv_tk_trace_on

167 /* Process model interrupt handling procedure */

168 void
169 cp_fddi_mac ()
170 {
171     /* Packets and ICI's */
172     Packet* mac_frame_ptr;
173     Packet* pdu_ptr;
174     Packet* pkptr;
175     Packet* data_pkptr;
176     Ici* ici_ptr;

177     /* Packet Fields and Attributes */
178     int req_pri, svc_class, req_tk_class;
179     int frame_control, src_addr, dest_addr;
180     int pk_len, pri_level;

181     /* Token - Related */

```

```

182 int tk_usable, res_station, tk_class;
183 int current_tk_class;
184 double accum_sync;

185 /* Timer - Related */
186 double tx_time, timer_remaining, accum_bandwidth;
187 double tht_value;

188 /* Miscellaneous */
189 int i;
190 int spawn_station, phy_arrival;
191 char error_string [512];
192 int num_frames_sent, num_bits_sent;

193 /* 26DEC93: loop management variables, used in RCV_TK */
194 /* and ENCAP states. -Nix */
195 int     NUM_PRIOS;
196 int     punt;
197 int     q_check;

198 FSM_ENTER (cp_fddi_mac)

199 FSM_BLOCK_SWITCH
200 {
201 /*-----*/
202 /** state (INIT) enter executives **/
203 FSM_STATE_ENTER_FORCED (0, state0_enter_exec, "INIT")
204 {
205 /* Obtain the station's address . This is an attribute */
206 /* of this process. Addressing is simplified by */
207 /* simply using integers, and only one mode. */
208 /* This mode is 16 bit addressing unless the */
209 /* packet format 'fddi_mac_fr' is modified. */
210 my_objid = op_id_self(); /* 29DEC93 */
211 op_ima_obj_attr_get (my_objid, "station_address", &my_address);

212 /* Register the station's object id in a global table. */
213 /* This table is used by the mechanism which improves */
214 /* simulation efficiency by 'jumping over' idle periods */
215 /* rather than circulating an unusable token. */
216 fddi_station_register (my_address, my_objid);

217 /* Obtain the station latency for tokens and frames. */
218 /* Default value is set at 100 nanoseconds. */
219 Fddi_St_Latency = 100.0e-09;
220 op_ima_sim_attr_get (OPC_IMA_DOUBLE, "station_latency", &Fddi_St_Latency);

```

```

221 /* Obtain the propagation delay separating stations. */
222 /* This value is given in seconds with default value 3.3 microseconds. */
223 Fddi_Prop_Delay = 3.3e-06;
224 op_ima_sim_attr_get (OPC_IMA_DOUBLE, "prop_delay", &Fddi_Prop_Delay);

225 /* Derive the Delay for a 'hop' of a freely circulating packet. */
226 Fddi_Tk_Hop_Delay = Fddi_Prop_Delay + Fddi_St_Latency;

227 /* The T_Pri [] state variable array supports priority */
228 /* assignments on a station by station basis by */
229 /* establishing a correspondence between integer priority */
230 /* levels assigned to frames and the maximum values of the */
231 /* token holding timer (THT) which would allow packets to be */
232 /* sent. Eight levels are supported here, but this can easily */
233 /* be changed by redimensioning the priority array. */
234 /* By default all levels are identical here, allowing */
235 /* any frame to make use of the token, so that in fact */
236 /* priority levels are not used in the default case. */

237 /* 01JAN94: (8-i) is a quick attempt to impart different weighting */
238 /* scales on each priority level, and is not necessarily realistic.-Nix */
239 /* Be aware of integer-double arithmetic conflicts ie, 1/8 = 0. -Nix */

240 op_ima_obj_attr_get(my_objid, "T_Req", &T_Req);
241 for (i = 0; i < 8; i++)
242 {
243     T_Pri[i] = ((double)(i + 1.0)/8.0) * Fddi_T_Opr;
244     /* printf("MAC INIT: T_Pri[%d] is %lf; */
245     /* Fddi_T_Opr is %lf\n", i, T_Pri[i], Fddi_T_Opr); */
246 }

247 /* Create the token holding timer (THT) used to restrict the */
248 /* asynchronous bandwidth consumption of the station */
249 THT = fddi_timer_create ();

250 /* Create the token rotation timer (TRT) used to measure the */
251 /* rotations of the token, detect late tokens and initialize */
252 /* the THT timer before asynchronous transmissions. */
253 TRT = fddi_timer_create ();

254 /* Set the TRT timer to expire in one TTRT */
255 TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);
256
257 /* Initialize the Late_Ct variable which keeps track. */
258 /* of the number of TRT expirations. */
259 Late_Ct = 0;

260 /* initially the ring operates in nonrestricted mode */
261 restricted = 0;

```

```

262 /* Create an Interface Control Information structure */
263 /* to use when delivering received frames to the LLC. */
264 to_llc_ici_ptr = op_ici_create ("fddi_mac_ind");

265 /* The 'tk_registered' variable indicates if the station */
266 /* has registered its intent to use the token. */
267 tk_registered = 0;

268 /* Determine if the model is to make use of the token */
269 /* 'acceleration' mechanism. If not, every passing of the */
270 /* token will be explicitly modeled, leading to large */
271 /* number of events being scheduled when the ring is idle */
272 /* (i.e., no stations have data to send). */
273 op_ima_sim_attr_get (OPC_IMA_INTEGER, "accelerate_token",
274 &Fddi_Tk_Accelerate);

275 /* Obtain the synchronous bandwidth assigned */
276 /* to this station. It is expressed as a */
277 /* percentage of TTRT, and then converted to seconds */
278 op_ima_obj_attr_get (my_objid, "sync bandwidth", &sync_pc);
279 sync_bandwidth = sync_pc * Fddi_T_Opr;

280 /* Only one station in the ring is selected to */
281 /* introduce the first token. Test if this station is it. */
282 /* If so, set the 'spawn_token' flag. */
283 op_ima_sim_attr_get (OPC_IMA_INTEGER, "spawn station", &spawn_station);
284 spawn_token = (spawn_station == my_address);
285 /* If the station is to spawn the token, create */
286 /* the packet which represents the token. */
287 /* 14APR94 :the bridges will spawn token in both rings */
288 /* -Karayakaylar */
289 spawn_token = 1;
290 if (spawn_token)
291 {
292 tk_pkptr = op_pk_create_fmt ("fddi_mac_tk");

293 /* assign its frame control field */
294 op_pk_nfd_set (tk_pkptr, "fc", FDDI_FC_TOKEN);

295 /* the first token issued is non-restricted */
296 op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);

297 /* The transition will be made into the ISSU_TK */
298 /* state where the tk_usable variable is used. */
299 /* In case any data has been generated, prset */
300 /* this variable to one. */

```

```

301 tk_usable = 1;
302 }

303 /* When sending packets the variable accum_bandwidth is */
304 /* used as a scheduling base. Init this value to zero. */
305 /* This statement is required in case this is the spawning */
306 /* station, and the next state entered is ISSUE_TK */
307 accum_bandwidth = 0.0;

308 }

309 /** state (INIT) exit executives **/
310 FSM_STATE_EXIT_FORCED (0, state0_exit_exec, "INIT")
311 {
312 }

313 /** state (INIT) transition processing **/
314 FSM_INIT_COND (spawn_token)
315 FSM_DFLT_COND
316 FSM_TEST_LOGIC ("INIT")

317 FSM_TRANSIT_SWITCH
318 {
319 FSM_CASE_TRANSIT (0, 2, state2_enter_exec, ;)
320 FSM_CASE_TRANSIT (1, 1, state1_enter_exec, ;)
321 }
322 /*-----*/

323 /** state (IDLE) enter executives **/
324 FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "IDLE")
325 {
326 }

327 /** blocking after enter executives of unforced state. **/
328 FSM_EXIT (3, cp_fddi_mac)

329 /** state (IDLE) exit executives **/
330 FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "IDLE")
331 {
332 /* Determine if a trace is activated for the FDDI model */
333 tk_trace_on = op_prg_odb_ltrace_active ("fddi_tk");

```

```

334 /* Trap packets arriving from physical layer so that their */
335 /* FC field can be extracted before evaluating conditions */
336 if (op_intrpt_type () == OPC_INTRPT_STRM && op_intrpt_strm () != FDDI_LLC_STRM_IN)
337 {
338 /* Acquire the arriving packet. */
339 pkptr = op_pk_get (FDDI_PHY_STRM_IN);

340 /* Determine the type of packet by extracting */
341 /* the frame control field. */
342 op_pk_nfd_get (pkptr, "fc", &frame_control);

343 /* Physical layer arrival flag is set. */
344 phy_arrival = 1;
345 }
346 else{
347 /* The interrupt is not due to a physical layer arrival. */
348 phy_arrival = 0;

349 /* If the interrupt is a remote interrupt with specified code, it signifies */
350 /* the reinsertion of the token into the ring after an idle period. This only */
351 /* occurs if the token acceleration mechanism is active. */
352 if (op_intrpt_type () == OPC_INTRPT_REMOTE && op_intrpt_code () == FDDIC_TK_INJECT)
353 {
354 /* create a new token */
355 tk_pkptr = op_pk_create_fmt ("fddi_mac_tk");

356 /* assign its frame control field */
357 op_pk_nfd_set (tk_pkptr, "fc", FDDI_FC_TOKEN);

358 /* the token is non-restricted */
359 op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);

360 /* insert it into the ring */
361 op_pk_send (tk_pkptr, FDDI_PHY_STRM_OUT);
362 }
363 }

}

364 /** state (IDLE) transition processing **/
365 FSM_INIT_COND (TK_RECEIVED)
366 FSM_TEST_COND (RC_FRAME)
367 FSM_TEST_COND (TRT_EXPIRE)
368 FSM_TEST_COND (FRAME_ARRIVAL)
369 FSM_DFLT_COND
370 FSM_TEST_LOGIC ("IDLE")

```

```

371 FSM_TRANSIT_SWITCH
372 {
373 FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;)
374 FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ;)
375 FSM_CASE_TRANSIT (2, 7, state7_enter_exec, ;)
376 FSM_CASE_TRANSIT (3, 8, state8_enter_exec, ;)
377 FSM_CASE_TRANSIT (4, 1, state1_enter_exec, ;)
378 }
379 /*-----*/

380 /** state (ISSUE_TK) enter executives **/
381 FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "ISSUE_TK")
382 {
383 /* If the token is sent without having been used, and the station */
384 /* has no data to send, then indicate this fact to the */
385 /* token acceleration mechanism which may have an */
386 /* opportunity to block the token. */
387 if (!tk_usable && op_q_stat (OPC_QSTAT_PKSIZE) == 0.0)
388 {
389 /* Note that if the token cannot be blocked, */
390 /* this procedure will forward the token physically. */
391 fddi_tk_indicate_no_data (tk_pkptr, my_address, accum_bandwidth);
392 }
393 else{
394 if (tk_trace_on == OPC_TRUE)
395 {
396 sprintf (str0, "Issuing token. accum_bw (%.9f), prop_del (%.9f)",
397 accum_bandwidth, Fddi_Prop_Delay);
398 op_prg_odb_print_major (str0, OPC_NIL);
399 }

400 /* Send out the token packet using the accumulated */
401 /* consumed bandwidth as a scheduling base. */
402 /* In the case of the initial spawning of the token */
403 /* this will be zero; otherwise this variable will */
404 /* reflect the bandwidth consumed since the last capture */
405 /* of the usable token. Propagation delay is also accounted for. */
406 op_pk_send_delayed (tk_pkptr, FDDI_PHY_STRM_OUT,
407 accum_bandwidth + Fddi_Prop_Delay);
408 }
409 }

410 /** state (ISSUE_TK) exit executives **/
411 FSM_STATE_EXIT_FORCED (2, state2_exit_exec, "ISSUE_TK")
412 {
413 }

```

```

414 /** state (ISSUE_TK) transition processing **/
415 FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
416 /*-----*/

417 /** state (RCV_TK) enter executives **/
418 FSM_STATE_ENTER_FORCED (3, state3_enter_exec, "RCV_TK")
419 {
420 /* The arriving packet, when received in the IDLE state */
421 /* is placed in the variable 'pkptr'. Since it is now */
422 /* known that it is a token, it can be placed in 'tk_pkptr. */
423 tk_pkptr = pkptr;
424
425 /* Load the token's class into the temporary variable 'tk_class.' */
426 op_pk_nfd_get (pkptr, "class", &tk_class);

427 /* If the token is restricted, determine for which station. */
428 if (tk_class == FDDI_TK_RESTRICTED)
429 {
430 /* Place the station address in the variable 'res_station' */
431 /* which may factor in to the determination of token usability. */
432 op_pk_nfd_get (tk_pkptr, "res_station", &res_station);
433 }

434 /* Determine if the token is usable: */
435 /* assume by default that it is not */
436 /* Subsequent conditions may override this. */
437 tk_usable = 0;

438 /* The token can only be usable if there are frames enqueued */
439 /* 27DEC93: the entire bank of subqueues must be checked, */
440 /* starting at the highest priority (corresponding to */
441 /* synchronous traffic), and stopping when a packet is */
442 /* found. Then the loop is broken. -Wix */
443 NUM_PRIOS = 9;
444 for (i = NUM_PRIOS - 1; i > -1; i--)
445 {
446 if (op_subq_stat (i, OPC_QSTAT_PKSIZE) > 0.0)
447 {
448 /* examine the attributes of the packet at the */
449 /* head of the queue. */
450 fddi_load_frame_attr (&dest_addr, &svc_class, &pri_level);

451 /* If synchronous data is queued, the token is */
452 /* necessarily usable, regardless of timing conditions. */
453 if (svc_class == FDDI_SVC_SYNC)
454 {

```



```

455 tk_usable = 1;
456 break;
457 }
458 else{
459 /* Otherwise, if asynchronous data is queued, it must */
460 /* meet several criteria for the token to be usable. */

461 /* The token is only usable only if it is early. */
462 if (Late_Ct == 0)
463 {
464 /* The token's class must be nonrestricted, unless */
465 /* this station is involved in the restricted transfer. */
466 if (tk_class == FDDI_TK_NONRESTRICTED ||
467 res_station == my_address ||
468 restricted)
469 {
470 /* Test the frame's priority assignment against the current TRT */
471 /* This test uses the priority indirection table T_Pri */
472 /* so that only packets whose T_Pri [pri_level] exceeds */
473 /* the TRT can be transmitted. In other words, by */
474 /* assigning lower values to T_Pri for a given priority */
475 /* level, packets of that level will be further restricted */
476 /* from using the ring bandwidth. */
477 if (T_Pri [pri_level] >= fddi_timer_value (TRT))
478 {
479 tk_usable = 1;
480 break;
481 }
482 }
483 }
484 }
485 } /* closes the "if (op_subq_stat (OPC_QSTAT_PKSIZE) > 0.0" statment */
486 } /* closes the "for" loop */

487 /* If the token is usable, timers must be readjusted. */
488 if (tk_usable)
489 {
490 /* The timer adjustment depends on whether the token is early or late. */
491 if (Late_Ct == 0)
492 {
493 /* Transfer the contents of TRT into THT. */
494 fddi_timer_copy (TRT, THT);

495 /* Disable the THT timer. */
496 fddi_timer_disable (THT);

497 /* Reset TRT to time the next rotation. */
498 op_ev_cancel (TRT_handle);
499 TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);

```

```

500 }
501 else{
502 /* If the token is late, set the THT to its expired */
503 /* value, and disable it. This will prevent any */
504 /* asynchronous transmissions from occurring. */
505 fddi_timer_set_value (THT, Fddi_T_Opr);
506 fddi_timer_disable (THT);

507 /* clear the Late token counter (note that TRT is not modified, */
508 /* so that less than a full TTRT remains before TRT expires again. */
509 Late_Ct = 0;
510 }
511 }

512 /* If the token is not usable, different adjustments are made. */
513 else{
514 /* Again, the adjustments depend on the lateness of the token */
515 if (Late_Ct == 0)
516 {
517 /* If the token is not late, the TRT is reset to time the next rotation. */
518 op_ev_cancel (TRT_handle);
519 TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);
520 }
521 else{
522 /* clear the Late token counter (note that TRT is not modified, */
523 /* so that less than a full TTRT remains before TRT expires again. */
524 Late_Ct = 0;
525 }

526 /* also, account for the time needed by the token */
527 /* to traverse the station, since it is about to be sent. */
528 /* Note: station latency is not inclusive of token */
529 /* transmission time, but only of the time required to */
530 /* process and repeat the token's symbols. */
531 accum_bandwidth = Fddi_St_Latency;
532 }

533 }

534 /** state (RCV_TK) exit executives **/
535 FSM_STATE_EXIT_FORCED (3, state3_exit_exec, "RCV_TK")
536 {
537 }

538 /** state (RCV_TK) transition processing **/
539 FSM_INIT_COND (tk_usable)
540 FSM_DFLT_COND

```

```

541 FSM_TEST_LOGIC ("RCV_TK"),

542 FSM_TRANSIT_SWITCH
543 {
544 FSM_CASE_TRANSIT (0, 9, state9_enter_exec, ;)
545 FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;)
546 }
547 /*-----*/

548 /** state (FR_RCV) enter executives **/
549 FSM_STATE_ENTER_FORCED (4, state4_enter_exec, "FR_RCV")
550 {
551 /* A frame has been received from the physical layer. Note that */
552 /* at this time, only the leading edge of the frame has arrived. */

553 /* Extract the frame's source address (this will be used to */
554 /* determine whether or not to strip the frame from the ring). */
555 op_pk_nfd_get (pkptr, "src_addr", &src_addr);

556 }

557 /** state (FR_RCV) exit executives **/
558 FSM_STATE_EXIT_FORCED (4, state4_exit_exec, "FR_RCV")
559 {
560 }

561 /** state (FR_RCV) transition processing **/
562 FSM_INIT_COND (STRIP)
563 FSM_DFLT_COND
564 FSM_TEST_LOGIC ("FR_RCV")

565 FSM_TRANSIT_SWITCH
566 {
567 FSM_CASE_TRANSIT (0, 5, state5_enter_exec, ;)
568 FSM_CASE_TRANSIT (1, 6, state6_enter_exec, ;)
569 }
570 /*-----*/

571 /** state (FR_STRIP) enter executives **/
572 FSM_STATE_ENTER_FORCED (5, state5_enter_exec, "FR_STRIP")
573 {
574 /* Destroy the frame which has now circulated the entire ring. */
575 op_pk_destroy (pkptr);
576 }

```

```

577 /** state (FR_STRIP) exit executives **/
578 FSM_STATE_EXIT_FORCED (5, state5_exit_exec, "FR_STRIP")
579 {
580 }

581 /** state (FR_STRIP) transition processing **/
582 FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
583 /*-----*/

584 /** state (FR_REPEAT) enter executives **/
585 FSM_STATE_ENTER_FORCED (6, state6_enter_exec, "FR_REPEAT")
586 {
587 /* Extract the destination address of the frame. */
588 op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

589 /* If the frame is for this station, make a copy */
590 /* of the frame's data field and forward it to */
591 /* the higher layer. */
592 /* 14APR94 : In order to send the frames which are */
593 /* addressed to the remote lan, check the address database */
594 /* of remote lan. Frames addressed to the remote lan shouldn't */
595 /* be repeated in the local ring -- This is a simple forwarding */
596 /* decision algorithm, one of the bridge's function */
597 /* - Karayakaylar */
598 if((dest_addr == my_address)|| (dest_addr > my_address))
599 {
600 /* record total size of the frame (including data) */
601 pk_len = op_pk_total_size_get (pkptr);

602 /* decapsulate the data contents of the frame */
603 /* 29JAN94: a new field, "pri", has been added to */
604 /* the fddi_llc_fr packet format in the Parameters */
605 /* Editor, so that output statistics can be */
606 /* generated by class and priority. -Nix */
607 op_pk_nfd_get (pkptr, "info", &data_pkptr);
608 op_pk_nfd_get (pkptr, "pri", &pri_level);

609 /* The source and destination address are placed in the */
610 /* LLC's ICI before delivering the frame's contents. */
611 op_ici_attr_set (to_llc_ici_ptr, "src_addr", src_addr);
612 op_ici_attr_set (to_llc_ici_ptr, "dest_addr", dest_addr);
613 op_ici_install (to_llc_ici_ptr);

614 /* Because, as noted in the FR_RCV state, only the */
615 /* frame's leading edge has arrived at this time, the */
616 /* complete frame can only be delivered to the higher */
617 /* layer after the frame's transmission delay has elapsed. */
618 /* (since decapsulation of the frame data contents has occurred, */

```

```

619 /* the original MAC frame length is used to calculate delay) */
620 tx_time = (double) pk_len / FDDI_TX_RATE;
621 op_pk_send_delayed (data_pkptr, FDDI_LLC_STRM_OUT, tx_time);

622 /* Note that the standard specifies that the original */
623 /* frame should be passed along until the originating station */
624 /* receives it, at which point it is stripped from the ring. */
625 /* However, in the simulation model, there is no interest */
626 /* in letting the frame continue past its destination unless */
627 /* group addresses are used, so that the same frame could be */
628 /* destined for several stations. Here the frame is stripped */
629 /* for efficiency as it reaches the destination; if the model */
630 /* is modified to include group addresses, this should be changed */
631 /* so that the frame is copied and the original repeated. */
632 /* Logic is already present for stripping the frame at the origin. */
633 op_pk_destroy (pkptr);
634 }
635 /* 14APR94 : the frames belong to this ring should be repeated. */
636 /* Thus, local traffic is constrained.-- This is filtering decision */
637 /* One of the bridge's function - Karayakaylar */
638 else{
639 /* Repeat the original frame on the ring and account for */
640 /* the latency through the station and the propagation delay */
641 /* for a single hop. */
642 /* (Only the originating station can strip the frame). */
643 op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT,
644 Fddi_St_Latency + Fddi_Prop_Delay);

645 }
646 }

647 /** state (FR_REPEAT) exit executives **/
648 FSM_STATE_EXIT_FORCED (6, state6_exit_exec, "FR_REPEAT")
649 {
650 }

651 /** state (FR_REPEAT) transition processing **/
652 FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
653 /*-----*/

654 /** state (TRT_EXP) enter executives **/
655 FSM_STATE_ENTER_FORCED (7, state7_enter_exec, "TRT_EXP")
656 {
657 /* The timer is reset and allowed to continue running. */
658 TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);

659 /* The late token counter is incremented. This will */

```

```

660 /* prevent this station from making any asynchronous */
661 /* transmissions when it next captures the token. */
662 Late_Ct++;

663 }

664 /** state (TRT_EXP) exit executives **/
665 FSM_STATE_EXIT_FORCED (7, state7_exit_exec, "TRT_EXP")
666 {
667 }

668 /** state (TRT_EXP) transition processing **/
669 FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
670 /*-----*/

671 /** state (ENCAP) enter executives **/
672 FSM_STATE_ENTER_FORCED (8, state8_enter_exec, "ENCAP")
673 {
674 /* A frame has arrived from a higher layer; place it in 'pdu_ptr'. */
675 pdu_ptr = op_pk_get (op_intrpt_strm ());

676 /* Also get the interface control information */
677 /* associated with the new frame. */
678 ici_ptr = op_intrpt_ici ();
679 if (ici_ptr == OPC_NIL)
680 {
681 sprintf (error_string, "Simulation aborted; error in object (%d)",
682 op_id_self ());
683 op_sim_end (error_string, "fddi_mac: required ICI not received", " ", " ");
684 }

685 /* Extract the requested service class */
686 /* (e.g. synchronous or asynchronous). */
687 if (op_ici_attr_exists (ici_ptr, "svc_class"))
688 op_ici_attr_get (ici_ptr, "svc_class", &svc_class);
689 else svc_class = FDDI_SVC_ASYNC;

690 /* Extract the destination address. */
691 op_ici_attr_get (ici_ptr, "dest_addr", &dest_addr);

692 /* Extract the original source address from ICI :16APR94 */
693 op_ici_attr_get (ici_ptr, "src_addr", &orig_src_addr);

694 /* If the frame is asynchronous, the priority and */
695 /* requested token class parameter may be specified. */
696 if (svc_class == FDDI_SVC_ASYNC)

```

```

697 {
698 /* Extract the requested priority level. */
699 if (op_ici_attr_exists (ici_ptr, "pri"))
700 op_ici_attr_get (ici_ptr, "pri", &req_pri);
701 else req_pri = 0;

702 /* Extract the token class (restricted or non-restricted). */
703 if (op_ici_attr_exists (ici_ptr, "tk_class"))
704 op_ici_attr_get (ici_ptr, "tk_class", &req_tk_class);
705 else req_tk_class = FDDI_TK_NONRESTRICTED;
706 }

707 /* Check for the default ICI values; if they are not present */
708 /* compose the frame: 21APR94 */
709 if (dest_addr != orig_src_addr){

710 /* Compose a mac frame from all these elements. */
711 mac_frame_ptr = op_pk_create_fmt ("fddi_mac_fr");
712 op_pk_nfd_set (mac_frame_ptr, "svc_class", svc_class);
713 op_pk_nfd_set (mac_frame_ptr, "dest_addr", dest_addr);
714 /*op_pk_nfd_set (mac_frame_ptr, "src_addr", my_address);*/
715 /* here original source address should be kept in mac frame : 16APR94 */
716 op_pk_nfd_set (mac_frame_ptr, "src_addr", orig_src_addr);
717 op_pk_nfd_set (mac_frame_ptr, "info", pdu_ptr);
718 printf("\ndest_addr = %5d\n", dest_addr);
719 printf("orig_src_addr = %5d\n", orig_src_addr);

720 if (svc_class == FDDI_SVC_ASYNC)
721 {
722 op_pk_nfd_set (mac_frame_ptr, "tk_class", req_tk_class);
723 op_pk_nfd_set (mac_frame_ptr, "pri", req_pri);
724 }

725 /* 04JAN94: if the frame is synchronous, assign it a separate */
726 /* priority so that it may be assigned its own subqueue, and */
727 /* thereby be assigned its own probe for monitoring. -Nix */
728 /* */
729 if (svc_class == FDDI_SVC_SYNC)
730 {
731 op_pk_nfd_set (mac_frame_ptr, "pri", 8);
732 }

733 /* Assign the frame control field, which in the model */
734 /* is used to distinguish between tokens and ordinary */
735 /* frames on the ring. */
736 op_pk_nfd_set (mac_frame_ptr, "fc", FDDI_FC_FRAME);

737 /* Enqueue the frame at the tail of the queue. */
738 /* 27DEC93: at the tail of the prioritized queue. */
739 /* 04JAN94: must distinguish between synch & asynch. */
740 if (svc_class == FDDI_SVC_ASYNC)
741 {

```

```

742   op_subq_pk_insert (req_pri, mac_frame_ptr, OPC_QPOS_TAIL);
743   }
744   if (svc_class == FDDI_SVC_SYNC)
745   {
746     op_subq_pk_insert (8, mac_frame_ptr, OPC_QPOS_TAIL);
747   }

748   /* if this station has not yet registered its intent to */
749   /* use the token, it may do so now since it has data to send */
750   if (!tk_registered)
751   {
752     fddi_tk_register ();
753     tk_registered = 1;
754   }

755   /* end of if(dest_addr != orig_src_addr) statement */

}

756   /** state (ENCAP) exit executives **/
757   FSM_STATE_EXIT_FORCED (8, state8_exit_exec, "ENCAP")
758   {
759   }

760   /** state (ENCAP) transition processing **/
761   FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
762   /*-----*/

763   /** state (TX_DATA) enter executives **/
764   FSM_STATE_ENTER_FORCED (9, state9_enter_exec, "TX_DATA")
765   {
766     /* In this state, frames are transmitted until the */
767     /* token is no longer usable. Frames are taken from */
768     /* the single input queue in FIFO order. */

769     /* Reset the accumulator used to keep track of bandwidth */
770     /* consumed by the transmissions. Because all the transmissions */
771     /* are scheduled to happen at the appropriate times, but */
772     /* these schedulings occur instantly, this accumulator serves */
773     /* as the scheduling base for the transmissions. */
774     /* In other words, each successively transmitted frame */
775     /* is delayed relative to the previous one by the time which */
776     /* the latter took to send. At the end of transmission (e.g. */
777     /* when the token is no longer usable), this accumulator */

```



```

779 /* serves to delay the forwarding of the token. */
780 accum_bandwidth = 0.0;

781 /* Note that, because all transmissions are */
782 /* scheduled, the value of the THT timer will not progress */
783 /* between shcedulings (these all happen in zero time), and so */
784 /* the variable 'tht_value' is used to emulate the timer's progress. */
785 tht_value = fddi_timer_value (THT);

786 /* ***** */
787 /* 30MAR94: print T_Pri[i]. THT data */
788 /* for (i = 0; i < 8; i++) */
789 /* { */
790 /* printf("TX_DATA: T_Pri[%d] = %d, THT = %d, Fddi_T_Opr = %d\n", i, T_Pri[i], tht_value */
791 /* } */

792 /* Reset an accumulator which reflects the consumed */
793 /* synchronous bandwidth. */
794 accum_sync = 0.0;

795 /* Reset counters for transmitted frames and bits. */
796 num_frames_sent = 0;
797 num_bits_sent = 0;

798 /* The transmission sequence must end if the input queue */
799 /* becomes exhausted. Other termination conditions are */
800 /* embedded in the loop. */
801 /* 27DEC93: modify the loop to accomodate subqueue structure. */
802 /* A "for" loop is imposed over the original "while" loop. */
803 /* First, reset the break marker, "punt". -Wix */
804 punt = 0;
805 for (i = NUM_PRIOS - 1; i > -1; i--)
806 {
807 while (op_subq_stat (i, OPC_QSTAT_PKSIZE) > 0.0)
808 {
809 /* Remove the next frame for transmission. */
810 pkptr = op_subq_pk_remove (i, OPC_QPOS_HEAD);
811
812 /* Obtain the frame's service class. */
813 op_pk_nfd_get (pkptr, "svc_class", &svc_class);
814
815 /* Synchronous and asynchronous frames are treated differently. */
816 if (svc_class == FDDI_SVC_SYNC)
817 {
818 /* Obtain the frame's length, and compute */
819 /* the time required to transmit it. */
820 pk_len = op_pk_total_size_get (pkptr);
821 tx_time = (double) pk_len / FDDI_TX_RATE;

822 /* Check if synchronous bandwidth allocation for this */

```

```

823 /* station would be exceeded if the transmission were to occur. */
824 if (accum_sync + tx_time > sync_bandwidth)
825 {
826 /* The frame could not be sent without exceeding */
827 /* the allocated synchronous bandwidth, */
828 /* so it is replaced on the queue. */
829 /* 27DEC93: in this case, 1 is the highest priority, */
830 /* which is reserved for synchronous traffic. -Nix */
831 op_subq_pk_insert (i, pkptr, OPC_QPOS_HEAD);

832 /* Exit the transmission loop since the frame */
833 /* transmission request cannot be honored. */
834 punt = 1;
835 break;
836 }
837 else{
838 /* Send the frame into the ring after other frames have completed. */
839 /* Also, account for its proagation delay; because the token propagation */
840 /* delay and the frame propagation delay must be consistent, and the */
841 /* token propagation delay is specified as a ring parameter (i.e., stations */
842 /* are assumed to be equally spaced), the ring is intended to run with */
843 /* the "delay" attributes of point-to-point links set at zero. */
844 op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT, accum_bandwidth + Fddi_Prop_Delay);

845 /* increase the consumed bandwidth to reflect this */
846 /* transmission. Also increase synchronous consumption. */
847 accum_bandwidth += tx_time;
848 accum_sync += tx_time;

849 /* Increase counters for transmitted bits and frames. */
850 num_frames_sent++;
851 num_bits_sent += pk_len;
852 }
853 }
854 else{
855 /* The request enqueued at the head of the queue is */
856 /* asynchronous. It may only be honored if the THT timer */
857 /* has not expired. */
858 if (tht_value >= Fddi_T_Opr)
859 {
860 /* replace the packet on the queue and exit the transmission loop. */
861 op_subq_pk_insert (i, pkptr, OPC_QPOS_HEAD);
862 punt = 1;
863 break;
864 }
865 else{
866 /* Obtain the priority assignment of the frame. */
867 op_pk_nfd_get (pkptr, "pri", &pri_level);

868 /* If the packet's assigned priority level */
869 /* is too low for it to be serviced, then exit the loop */

```

```

870 /* after replacing the packet in the queue. */
871 if (T_Pri [pri_level] < tht_value)
872 {
873   op_subq_pk_insert (i, pkptr, OPC_QPOS_HEAD);
874   punt = 1;
875 break;
876 }

877 /* Obtain the frame's length, and compute the time */
878 /* which would be required to transmit it. */
879 pk_len = op_pk_total_size_get (pkptr);
880 tx_time = (double) pk_len / FDDI_TX_RATE;

881 /* Determine the requested token class to be */
882 /* released after this frame is transmitted. */
883 op_pk_nfd_get (pkptr, "tk_class", &tk_class);

884 /* If the station is in restricted mode, then it may */
885 /* exit this mode if the class is now nonrestricted */
886 /* or if the restricted peer is not the addressee. */
887 if (restricted)
888 {
889   /* Determine the destination address for the new packet. */
890   op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

891   if (tk_class == FDDI_TK_NONRESTRICTED ||
892       res_peer != dest_addr)
893   {
894     /* Exit restricted mode */
895     restricted = 0;
896
897     /* Modify the token to reflect the mode change. */
898     op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);
899   }
900 }
901 else{
902   /* Determine the class of the current captured token. */
903   op_pk_nfd_get (tk_pkptr, "class", &current_tk_class);

904   /* When not in restricted mode, this mode may be entered */
905   /* if the passed packet has the appropriate token class requested, */
906   /* and the token is not already restricted. */
907   if (tk_class == FDDI_TK_RESTRICTED && current_tk_class != FDDI_TK_RESTRICTED)
908   {

909     /* Enter restricted mode. */
910     restricted = 1;

911     /* Store the address of the restricted peer station. */
912     op_pk_nfd_get (pkptr, "dest_addr", &res_peer);

```

```

913 /* Modify the token to reflect the mode change. */
914 op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_RESTRICTED);
915 op_pk_nfd_set (tk_pkptr, "res_station", res_peer);
916 }
917 }

918 /* Send the frame once previous transmissions have completed. */
919 /* Account for propagation delay as well. */
920 op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT, accum_bandwidth + Fddi_Prop_Delay);

921 /* Increment THT emulation variable, and consumed bandwidth accumulator. */
922 tht_value += tx_time;
923 accum_bandwidth += tx_time;

924 /* Increase counters for transmitted bits and frames. */
925 num_frames_sent++;
926 num_bits_sent += pk_len;
927 }
928 }
929 } /* closes the 'while' loop */
930 if (punt == 1) /* If the 'while' loop was broken, */
931 {
932     punt = 0; /* then reset the 'break' marker, */
933     break; /* and break out of the 'for' loop too. */
934 }
935 } /* closes the 'for' loop. */

936 /* Since the token is about to be sent, its transmission time */
937 /* must be reflected in the accumulated bandwidth. This is not */
938 /* done in the ISSUE_TK state because when the token is merely */
939 /* repeated, full transmission delay is not required, only */
940 /* a small delay for repeating. */
941 accum_bandwidth += FDDIC_TOKEN_TX_TIME;

942 /* If the station has no more data to send (synchronous or */
943 /* asynchronous), it should indicate this to the token acceleration */
944 /* mechanism by deregistering its interest in the token. */
945 /* 27DEC94: the original code must be modified to include a check */
946 /* of subqueues. -Nix */
947 q_check = 1;
948 for (i = NUM_PRIOS - 1; i < -1; i--)
949 {
950     if (op_subq_stat (i, OPC_QSTAT_PKSIZE) == 0.0)
951     {
952         q_check = 0;
953     }
954     else {
955         q_check = 1;
956         break;

```

```

957     }
958 }

959 if (tk_registered && q_check == 0)
960 {
961 tk_registered = 0;
962 fddi_tk_deregister ();
963 }

964 }

965 /** state (TX_DATA) exit executives **/
966 FSM_STATE_EXIT_FORCED (9, state9_exit_exec, "TX_DATA")
967 {
968 }

969 /** state (TX_DATA) transition processing **/
970 FSM_TRANSIT_FORCE (2, state2_enter_exec, ;)
971 /*-----*/

972 /** state (CLAIM) enter executives **/
973 FSM_STATE_ENTER_UNFORCED (10, state10_enter_exec, "CLAIM")
974 {
975 /* Obtain this station's object id which is used */
976 /* to access the station's attribute assignments. */
977 my_objid = op_id_self ();

978 /* Using the object id, obtain the ring id. */
979 /* The ring id is used by macros defined in the */
980 /* header block to obtain "ring-global" values, */
981 /* values shared by all stations on a ring. */
982 op_ima_obj_attr_get (my_objid, "ring_id", &ring_id);

983 /* Initialize global variable values. */
984 Fddi_Tk_Blocked = 0;
985 Fddi_Num_Stations = 0;
986 Fddi_Num_Registered = 0;

987 /* Using the object id, obtain the value of 'T_Req', */
988 /* the value of TTRT requested by this station. */
989 op_ima_obj_attr_get (my_objid, "T_Req", &T_Req);

990 /* 30MAR: workaround; Fddi_T_Opr is never initialized in the */
991 /* original code. -Nix */
992 Fddi_T_Opr = 500;

```

```

993 /* The lowest value of T_Req becomes T_Opr for the ring as a whole. */
994 if (T_Req < Fddi_T_Opr || Fddi_Claim_Start)
995 {
996 /* The T_Req for this station is lower than any other to date */
997 /* so it is installed in the T_Opr variable. */
998 Fddi_T_Opr = T_Req;

999 /* The flag indicating that the claim process is just */
1000 /* beginning may now be cleared. */
1001 Fddi_Claim_Start = 0;
1002 }

1003 /* Request a self interrupt from the Smulation Kernel at the current
1004 /* time so that after all stations have executed their claim states, */
1005 /* they can proceed with initializations. This is necessasary */
1006 /* because some initializations are based in the value of T_Opr */
1007 /* and it must therefore be known that all stations have settled */
1008 /* on a final value. */
1009 op_intrpt_schedule_self (op_sim_time (), 0);

1010 }

1011 /** blocking after enter executives of unforced state. **/
1012 FSM_EXIT (21,cp_fddi_mac)

1013 /** state (CLAIM) exit executives **/
1014 FSM_STATE_EXIT_UNFORCED (10, state10_exit_exec, "CLAIM")
1015 {
1016 }

1017 /** state (CLAIM) transition processing **/
1018 FSM_TRANSIT_FORCE (0, state0_enter_exec, ;)
1019 /*-----*/

1020 }

1021 FSM_EXIT (10,cp_fddi_mac)
1022 }

```

```

1023 void
1024 cp_fddi_mac_svar (prs_ptr,var_name,var_p_ptr)
1025 cp_fddi_mac_state *prs_ptr;
1026 char *var_name, **var_p_ptr;
1027 {

1028 FIN (cp_fddi_mac_svar (prs_ptr))

1029 *var_p_ptr = VOS_NIL;
1030 if (Vos_String_Equal ("ring_id" , var_name))
1031 *var_p_ptr = (char *) (&prs_ptr->sv_ring_id);
1032 if (Vos_String_Equal ("THT" , var_name))
1033 *var_p_ptr = (char *) (&prs_ptr->sv_THT);
1034 if (Vos_String_Equal ("T_Req" , var_name))
1035 *var_p_ptr = (char *) (&prs_ptr->sv_T_Req);
1036 if (Vos_String_Equal ("T_Pri" , var_name))
1037 *var_p_ptr = (char *) (prs_ptr->sv_T_Pri);
1038 if (Vos_String_Equal ("my_objid" , var_name))
1039 *var_p_ptr = (char *) (&prs_ptr->sv_my_objid);
1040 if (Vos_String_Equal ("spawn_token" , var_name))
1041 *var_p_ptr = (char *) (&prs_ptr->sv_spawn_token);
1042 if (Vos_String_Equal ("my_address" , var_name))
1043 *var_p_ptr = (char *) (&prs_ptr->sv_my_address);
1044 if (Vos_String_Equal ("orig_src_addr" , var_name))
1045 *var_p_ptr = (char *) (&prs_ptr->sv_orig_src_addr);
1046 if (Vos_String_Equal ("tk_pkptr" , var_name))
1047 *var_p_ptr = (char *) (&prs_ptr->sv_tk_pkptr);
1048 if (Vos_String_Equal ("sync_bandwidth" , var_name))
1049 *var_p_ptr = (char *) (&prs_ptr->sv_sync_bandwidth);
1050 if (Vos_String_Equal ("sync_pc" , var_name))
1051 *var_p_ptr = (char *) (&prs_ptr->sv_sync_pc);
1052 if (Vos_String_Equal ("restricted" , var_name))
1053 *var_p_ptr = (char *) (&prs_ptr->sv_restricted);
1054 if (Vos_String_Equal ("res_peer" , var_name))
1055 *var_p_ptr = (char *) (&prs_ptr->sv_res_peer);
1056 if (Vos_String_Equal ("tk_registered" , var_name))
1057 *var_p_ptr = (char *) (&prs_ptr->sv_tk_registered);
1058 if (Vos_String_Equal ("to_llc_ici_ptr" , var_name))
1059 *var_p_ptr = (char *) (&prs_ptr->sv_to_llc_ici_ptr);
1060 if (Vos_String_Equal ("tk_trace_on" , var_name))
1061 *var_p_ptr = (char *) (&prs_ptr->sv_tk_trace_on);

1062 FOOT;
1063 }

1064 void
1065 cp_fddi_mac_diag ()
1066 {

```

```

1057 /* Packets and ICI's */
1058 Packet* mac_frame_ptr;
1059 Packet* pdu_ptr;
1060 Packet* pkptr;
1061 Packet* data_pkptr;
1062 Ici* ici_ptr;

1073 /* Packet Fields and Attributes */
1074 int req_pri, svc_class, req_tk_class;
1075 int frame_control, src_addr, dest_addr;
1076 int pk_len, pri_level;

1077 /* Token - Related */
1078 int tk_usable, res_station, tk_class;
1079 int current_tk_class;
1080 double accum_sync;

1081 /* Timer - Related */
1082 double tx_time, timer_remaining, accum_bandwidth;
1083 double tht_value;

1084 /* Miscellaneous */
1085 int i;
1086 int spawn_station, phy_arrival;
1087 char error_string [512];
1088 int num_frames_sent, num_bits_sent;

1089 /* 26DEC93: loop management variables, used in RCV_TK */
1090 /* and ENCAP states. -Nix */
1091 int      NUM_PRIOS;
1092 int      punt;
1093 int      q_check;

1094 FIN (cp_fddi_mac_diag ())

1095 /* Print out values of timers, and late token counter. */
1096 /* Also print out data about restricted mode. */
1097 /* (This code may be executed by the simulation debugger */
1098 /* by invoking the command 'modprint'). */

1099 sprintf (str0, "Timers (count upwards): TRT (%.9g), THT (%.9g)",
1100 fddi_timer_value (TRT), fddi_timer_value (THT));
1101 sprintf (str1, "Late_ct (%d)", Late_Ct);
1102 op_prg_odb_print_major (str0, str1, OPC_NIL);

1103 if (restricted)
1104     sprintf (str0, "token is in restricted dialog with (%d)\n", res_peer);
1105 else sprintf (str0, "token is unrestricted\n");

```



```

1106 op_prg_odb_print_major (str0, OPC_NIL);

1107 FOUT;
1108 }

1109 void
1110 cp_fddi_mac_terminate ()
1111 {
1112 /* Packets and ICI's */
1113 Packet* mac_frame_ptr;
1114 Packet* pdu_ptr;
1115 Packet* pkptr;
1116 Packet* data_pkptr;
1117 Ici* ici_ptr;

1118 /* Packet Fields and Attributes */
1119 int req_pri, svc_class, req_tk_class;
1120 int frame_control, src_addr, dest_addr;
1121 int pk_len, pri_level;

1122 /* Token - Related */
1123 int tk_usable, res_station, tk_class;
1124 int current_tk_class;
1125 double accum_sync;

1126 /* Timer - Related */
1127 double tx_time, timer_remaining, accum_bandwidth;
1128 double tht_value;

1129 /* Miscellaneous */
1130 int i;
1131 int spawn_station, phy_arrival;
1132 char error_string [512];
1133 int num_frames_sent, num_bits_sent;

1134 /* 26DEC93: loop management variables, used in RCV_TK */
1135 /* and ENCAP states. -Wix */
1136 int      NUM_PRIOS;
1137 int      punt;
1138 int      q_check;

1139 FIN (cp_fddi_mac_terminate ())

```

```

1140 FOUT;
1141 }

1142 Compcode
1143 cp_fddi_mac_init (pr_state_pptr)
1144 cp_fddi_mac_state **pr_state_pptr;
1145 {
1146 static VosT_Cm_Obtype obtype = OPC_NIL;

1147 FIN (cp_fddi_mac_init (pr_state_pptr))

1148 if (obtype == OPC_NIL)
1149 {
1150 if (Vos_Catmem_Register ("proc state vars (cp_fddi_mac)",
1151 sizeof (cp_fddi_mac_state), Vos_Nop, &obtype) == VOSC_FAILURE)
1152 FRET (OPC_COMPCODE_FAILURE)
1153 }

1154 if ((*pr_state_pptr = (cp_fddi_mac_state) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
1155 FRET (OPC_COMPCODE_FAILURE)
1156 else
1157 {
1158 (*pr_state_pptr)->current_block = 20;
1159 FRET (OPC_COMPCODE_SUCCESS)
1160 }
1161 }

1162 /** The procedures defined in this section serve **/
1163 /** to simplify the code in the main body of the **/
1164 /** process model by providing primitives for timer **/
1165 /** manipulation.. **/

1166 static
1167 fddi_timer_disable (timer_ptr)
1168 FddiT_Timer* timer_ptr;
1169 {
1170 /* if the timer is already disabled, do nothing */
1171 if (timer_ptr->enabled)
1172 {
1173 /* disable the timer */
1174 timer_ptr->enabled = 0;

1175 /* reassign the accumulated time so far */
1176 timer_ptr->accum = op_sim_time () - timer_ptr->start_time;
1177 }
1178 }

```

```

1179 static
1180 fddi_timer_enable (timer_ptr)
1181 FddiT_Timer* timer_ptr;
1182 {
1183 /* if the timer is already enabled, simply return */
1184 if (!timer_ptr->enabled)
1185 {
1186 /* reenale the timer */
1187 timer_ptr->enabled = 1;

1188 /* set the start time to the current time */
1189 /* less the accumulated time so far */
1190 timer_ptr->start_time = op_sim_time () - timer_ptr->accum;
1191 }
1192 }

1193 static
1194 fddi_timer_expired (timer_ptr)
1195 FddiT_Timer* timer_ptr;
1196 {
1197 if (fddi_timer_remaining (timer_ptr) <= 0.0)
1198 return 1;
1199 else return 0;
1200 }

1201 static
1202 double
1203 fddi_timer_remaining (timer_ptr)
1204 FddiT_Timer* timer_ptr;
1205 {
1206 /* if the timer is enabled, update the accumulated time */
1207 if (timer_ptr->enabled)
1208 {
1209 timer_ptr->accum = op_sim_time () - timer_ptr->start_time;
1210 }

1211 /* return the timer remaining before expiration */
1212 /* a non-positive value indicates an expired timer */
1213 return (timer_ptr->target_accum - timer_ptr->accum);
1214 }

1215 static
1216 double
1217 fddi_timer_value (timer_ptr)
1218 FddiT_Timer* timer_ptr;
1219 {
1220 /* if the timer is enabled, update the accumulated time */
1221 if (timer_ptr->enabled)
1222 {
1223 timer_ptr->accum = op_sim_time () - timer_ptr->start_time;

```

```

1224     }

1225 return (timer_ptr->accum);
1226 }

1227 static
1228 fddi_timer_set_value (timer_ptr, value)
1229 FddiT_Timer* timer_ptr;
1230 double value;
1231 {
1232 timer_ptr->accum = value;
1233 }
1234
1235 static
1236 fddi_timer_copy (from_timer_ptr, to_timer_ptr)
1237 FddiT_Timer* from_timer_ptr;
1238 FddiT_Timer* to_timer_ptr;
1239 {
1240 Vos_Copy_Memory (from_timer_ptr, to_timer_ptr,
1241 sizeof (FddiT_Timer));
1242 }

1243 static
1244 fddi_timer_set (timer_ptr, duration)
1245 FddiT_Timer* timer_ptr;
1246 {
1247 /* clear out accumulated time */
1248 timer_ptr->accum = 0.0;

1249 /* assign the timer duration */
1250 timer_ptr->target_accum = duration;

1251 /* assign the current time */
1252 timer_ptr->start_time = op_sim_time ();

1253 /* enable the timer */
1254 timer_ptr->enabled = 1;
1255 }

1256 static
1257 FddiT_Timer*
1258 fddi_timer_create ()
1259 {
1260 FddiT_Timer* timer_ptr;

1260 /* allocate memory for a timer structure */
1261 timer_ptr = (FddiT_Timer*) malloc (sizeof (FddiT_Timer));

1262 /* initialize the timer in the disabled mode */
1263 fddi_timer_init (timer_ptr);

```

```

1264 /* return the timer's address */
1265 return (timer_ptr);
1266 }

1267 static
1268 fddi_timer_init (timer_ptr)
1269 FddiT_Timer* timer_ptr;
1270 {
1271 /* the timer is initially disabled */
1272 timer_ptr->enabled = 0;

1273 /* the accumulated time is zero */
1274 timer_ptr->accum = 0.0;

1275 /* the target accumulated time is infinite */
1276 timer_ptr->target_accum = VOS_DOUBLE_INFINITY;

1277 /* the start time is now */
1278 timer_ptr->start_time = op_sim_time ();
1279 }

1280 static
1281 fddi_station_register (address, objid)
1282 Objid objid;
1283 int address;
1284 {
1285 /* Fill an entry in the table which maps station */
1286 /* addresses to OPNET object ids */
1287 FIN (fddi_station_register (address, objid))

1288 Fddi_Address_Table [address] = objid;

1289 /* Keep track of total number of stations on the ring */
1290 Fddi_Num_Stations++;

1291 FOUT
1292 }

1293 static
1294 fddi_tk_register ()
1295 {
1296 /* Register the station's intent to use the token. */
1297 /* This should be done whenever an unregistered */
1298 /* station obtains new data to transmit. */
1299 FIN (fddi_tk_register ())

1300 /* increase the number of registered stations */
1301 Fddi_Num_Registered++;

1302 /* if the token is currently blocked, unblock it */
1303 if (Fddi_Tk_Blocked && Fddi_Tk_Accelerate)
1304 {

```

```

1305 fddiTk_unblock ();
1306 }

1307 FOUT
1308 }
1309 static
1310 fddiTk_deregister ()
1311 {
1312 /* Cancel the station's intent to use the token. */
1313 /* This should be done whenever a registered */
1314 /* station exhausts its transmittable data. */
1315 FIN (fddiTk_deregister ())

1316 /* decrease the number of registered stations */
1317 Fddi_Num_Registered--;

1318 FOUT
1319 }

1320 static
1321 fddiTk_indicate_no_data (token, address, delay)
1322 Packet* token;
1323 int address;
1324 double delay;
1325 {
1326 FIN (fddiTk_indicate_no_data (token, address, delay))

1327 /* The calling station is indicating that it has captured */
1328 /* the token, but has no data to send. If no other stations */
1329 /* have data to send either, the token may be blocked to gain */
1330 /* simulation efficiency. */
1331 if (Fddi_Num_Registered == 0 && Fddi_Tk_Accelerate)
1332 {
1333 fddiTk_block (token, address);
1334 }
1335 else{
1336 /* If the token cannot be blocked, send it into the ring. */
1337 op_pk_send_delayed (token, FDDI_PHY_STRM_OUT,
1337 delay + Fddi_Prop_Delay);
1338 }

1339 FOUT
1340 }

1341 static
1342 fddiTk_block (token, address)
1343 Packet* token;
1344 int address;
1345 {
1346 int i;

```

```

1347 FIN (fddiTk_block (token, address))

1348 /* Record the address of the blocking station and blocking time. */
1349 FddiTk_Block_Base_Time = op_sim_time ();
1350 FddiTk_Block_Base_Station = address;

1351 if (tk_trace_on == OPC_TRUE)
1352 {
1353     sprintf (str0, "Blocking Token: station (%d), time (%.9f)",
1354             FddiTk_Block_Base_Station, FddiTk_Block_Base_Time);
1355     op_prg_odb_print_major (str0, OPC_NIL);
1356 }

1357 /* Indicate that the token is blocked */
1358 FddiTk_Blocked = 1;

1359 /* discard the token packet; another one will be */
1360 /* created when the token is unblocked. */
1361 op_pk_destroy (token);

1362 /* Cancel TRT timers at all MAC interfaces; otherwise these */
1363 /* timers may continue to expitr during the idle period, */
1364 /* generating unnecessary events. */
1365 if (tk_trace_on == OPC_TRUE)
1366 {
1367     sprintf (str0, "Canceling timers for (%d) stations", Fddi_Num_Stations);
1368     op_prg_odb_print_major (str0, OPC_NIL);
1369 }

1370 for (i = 0; i < Fddi_Num_Stations; i++)
1371 {
1372     /* Retain the time at which the TRT would have expired; */
1373     /* this is used for calculations when the token is */
1374     /* reinjected into the ring. */
1375     Fddi_Trt_Exp_Time [i] = op_ev_time (Fddi_Trt_Handle [i]);

1376     /* Cancel the TRT expiration event. */
1377     op_ev_cancel (Fddi_Trt_Handle [i]);
1378 }

1379 FOUT
1380 }

1381 static
1382 fddiTk_unblock ()
1383 {
1384     double elapsed_time, firstTk_rx, lastTk_rx;
1385     double tk_lap_time, next_time, current_time;
1386     double dbl_num_hops, numTk_rx, floor (), ceil ();
1387     int i, num_hops, next_station;

```

```

1388 FIN (fddi_tk_unblock ())

1389 /* reset the blocking indicator */
1390 Fddi_Tk_Blocked = 0;

1391 /* Get the current time, used for many calculations below */
1392 current_time = op_sim_time ();

1393 if (tk_trace_on == OPC_TRUE)
1394 {
1395     sprintf (str0, "Unblocking token for ring (%d)", ring_id);
1396     op_prg_odb_print_major (str0, OPC_NIL);
1397 }

1397 /* For all stations on the ring, adjust TRT timer and Late_Ct flag. */
1398 for (i = 0; i < Fddi_Num_Stations; i++)
1399 {
1400     if (tk_trace_on == OPC_TRUE)
1401     {
1402         sprintf (str0, "adjusting state of station (%d)", i);
1403         op_prg_odb_print_minor ("", str0, OPC_NIL);
1404     }
1405     /* Calculate number of hops separating station i from block base station. */
1406     /* In special case where i is the base station, the token must run a full */
1407     /* lap before returning. */
1408     if (i != Fddi_Tk_Block_Base_Station)
1409     {
1410         num_hops = (i - Fddi_Tk_Block_Base_Station) % Fddi_Num_Stations;
1411         if (num_hops < 0)
1412             num_hops = Fddi_Num_Stations + num_hops;
1413     }
1414     else num_hops = Fddi_Num_Stations;

1415     /* Calculate first time at which token would have been received by station i. */
1416     /* Note that initial release of token from base station takes a different */
1417     /* amount of time than repeating of token by other stations. Thus, the first */
1418     /* hop is assumed, and the base time is augmented by the time required to */
1419     /* complete it. */
1420     first_tk_rx = Fddi_Tk_Block_Base_Time + FDDIC_TOKEN_TX_TIME + Fddi_Prop_Delay +
1421     (num_hops - 1) * Fddi_Tk_Hop_Delay;

1422     if (tk_trace_on == OPC_TRUE)
1423     {
1424         sprintf (str0, "station is (%d) hops from base", num_hops);
1425         sprintf (str1, "first receipt of token would be at (%.9f)", first_tk_rx);
1426         op_prg_odb_print_minor (str0, str1, OPC_NIL);
1427     }

1428     /* Case 1: the token would not yet have been received by station i. */
1429     if (first_tk_rx > current_time)

```



```

1430 {
1431 /* Case 1a: the TRT at station i would not yet have expired. */
1432 if (Fddi_Trt_Exp_Time [i] > current_time)
1433 {
1434 /* Late_Ct remains at its original value; only the TRT needs */
1435 /* to be started again, with the same expiration time. */
1436 TRT_SET (i, Fddi_Trt_Exp_Time [i])

1437 if (tk_trace_on == OPC_TRUE)
1438 {
1439 sprintf (str0, "Restoring TRT to previous exp. time (%.9f)", Fddi_Trt_Exp_Time [i]);
1440 op_prg_oddb_print_minor ("Token would not be received and TRT not expired", str0, OPC_NIL);
1441 }
1442 }
1443 /* Case 1b: the TRT at station i would have expired. */
1444 else
1445 {
1446 /* Late_Ct would have been set; also the timer would have been rescheduled */
1447 /* for an entire TTRT at the time of expiration. */
1448 Fddi_Late_Ct [i] = 1;
1449 TRT_SET (i, (Fddi_T_Opr + Fddi_Trt_Exp_Time [i]))

1450 if (tk_trace_on == OPC_TRUE)
1451 {
1452 sprintf (str0, "Restoring TRT to proper exp. time (%.9f)", Fddi_T_Opr + Fddi_Trt_Exp_Time [
1453 op_prg_oddb_print_minor ("Token would not be received and TRT would have expired", str0, OPC
1454 }
1455 }
1456 }

1457 /* Case 2: the token would have been received (perhaps more than once). */
1458 else
1459 {
1460 /* Calculate the number of times the token would have been received */
1461 /* not including the first receipt. */
1462 tk_lap_time = Fddi_Tk_Hop_Delay * Fddi_Num_Stations;
1463 num_tk_rx = floor ((current_time - first_tk_rx) / tk_lap_time);

1464 /* Calculate the latest time at which the token would have been received. */
1465 last_tk_rx = first_tk_rx + (num_tk_rx * tk_lap_time);
1466
1467 /* Clear Late_Ct and schedule timer to expire at last receipt of token */
1468 /* plus one full TTRT. */
1469 Fddi_Late_Ct [i] = 0;
1470 TRT_SET (i, (last_tk_rx + Fddi_T_Opr))

1471 if (tk_trace_on == OPC_TRUE)
1472 {
1473     sprintf (str0, "token received (%g) times, last receipt at (%.9f)",
1474 num_tk_rx + 1.0, last_tk_rx);
1475 sprintf (str1, "Restoring TRT to proper exp. time (%.9f)",

```

```

1476 last_tk_rx + Fddi_T_Opr);
1477 op_prg_odb_print_minor ("Token would have been received; Late_Ct is cleared",
1478 str1, str0, OPC_NIL);
1479 }
1480 }
1481 }

1482 /* compute the time since the token was blocked */
1483 elapsed_time = current_time - Fddi_Tk_Block_Base_Time;

1484 /* compute the number of hops completed on the ring. For the first hop */
1485 /* the token is transmitted directly, not repeated. For all remaining */
1486 /* hops, the delay is the station latency plus the propagation delay. */
1487 /* Thus, the first hop is assumed, and the remaining time for additional*/
1488 /* hops is computed beginning at the time where the token enters the */
1489 /* base station's downstream neighbor. */
1490 dbl_num_hops = 1.0 +
1491 (elapsed_time - FDDIC_TOKEN_TX_TIME - Fddi_Prop_Delay) / Fddi_Tk_Hop_Delay;

1492 /* If the token was unblocked in less time than it would have taken to */
1493 /* be fully transmitted by the base station, dbl_num_hops will be */
1494 /* negative. However, 1 full hop would still be required before the */
1495 /* token could be used, since the station had already committed to */
1496 /* issuing the token. Thus, the actual of number of hops should never */
1497 /* be less than 1. If it is, round it to 1. */
1498 if (dbl_num_hops < 1.0)
1499   dbl_num_hops = 1.0;
1500 else
1501 {
1502   /* In all other cases, round the number of hops up to the nearest */
1503   /* integer value. If already an integer, then leave as is. */
1504   dbl_num_hops = ceil (dbl_num_hops);
1505 }

1506 /* Obtain an integer equivalent of dbl_num_hops. */
1507 num_hops = dbl_num_hops;

1508 /* Based on the number of hops and the base station, compute the */
1509 /* next station where the token will appear. */
1510 next_station = (num_hops + Fddi_Tk_Block_Base_Station) % Fddi_Num_Stations;

1511 /* Compute the time at which the token will appear there. */
1512 /* Again, assume the first hop occurred, and measure time */
1513 /* from there forward. */
1514 next_time = Fddi_Tk_Block_Base_Time + (FDDIC_TOKEN_TX_TIME + Fddi_Prop_Delay) +
1515 (dbl_num_hops - 1.0) * Fddi_Tk_Hop_Delay;

1516 if (tk_trace_on == OPC_TRUE)
1517 {
1518   sprintf (str0, "Re-introducing token at station (%d), at time (%.9f)",
1519     next_station, next_time);

```

```

1520 op_prg_odb_print_minor (str0, OPC_NIL);
1521 }

1522 /* reinject the token at that station */
1523 fddi_tk_inject (next_station, next_time);

1524 FOUT
1525 }

1526 static
1527 fddi_tk_inject (address, arv_time)
1528 int address;
1529 double arv_time;
1530 {
1531 /* Re-insert the token into the ring after an idle period. */
1532 FIN (fddi_tk_inject (address, arv_time))

1533 /* The token is recreated and reinserted onto the ring */
1534 /* at the specified station which is not necessarily the */
1535 /* station now requesting the token. */
1536 /* The station which will reinsert the token is */
1537 /* asked to do so by means of a remote interrupt. */
1538 op_intrpt_schedule_remote (arv_time, FDDIC_TK_INJECT,
1539 Fddi_Address_Table [address]);

1540 FOUT
1541 }

1542 static
1543 fddi_load_frame_attrs (dest_addr_ptr, svc_class_ptr, pri_level_ptr)
1544 int *dest_addr_ptr, *svc_class_ptr, *pri_level_ptr;
1545 {
1546     int          NUM_PRIOS, i;    /* 26JAN94 */
1547 Packet *pkptr;

1548 FIN (fddi_load_frame_attrs (dest_addr_ptr, svc_class_ptr, pri_level_ptr))

1549 /* remove next packet in queue */
1550 /* 27DEC94: loop structure superimposed to handle a bank of subqueues. */
1551 /* Extract the packet with the highest priority, that is, the packet */
1552 /* at the head of the highest-numbered subqueue containing packets. */
1553 /* Note that the C language vector numbering convention numbers the */
1554 /* subqueues from 0 to 7, while FDDI convention is to number the */
1555 /* corresponding asynchronous priorities from 1 to 8. This is */
1556 /* reconciled in the statistical outputs available in the Analysis */
1557 /* Editor, where labels are assigned accordingly. Also note that */
1558 /* synchronous traffic is assigned priority 8 as an artifice to allow */
1559 /* routing through a separate subqueue, by which statistics may be */
1560 /* gathered for traffic by class and by priority. -Nix */
1561 NUM_PRIOS = 9;
1562 for (i = NUM_PRIOS - 1; i > -1; i--)

```

```

1563     {
1564         if (op_subq_stat (i, OPC_QSTAT_PKSIZE) > 0.0)
1565         {
1566             pkptr = op_subq_pk_remove (i, OPC_QPOS_HEAD);
1567             break;
1568         }
1569     }
1570 /* extract the fields of interest */
1571 op_pk_nfd_get (pkptr, "dest_addr", dest_addr_ptr);
1572 op_pk_nfd_get (pkptr, "svc_class", svc_class_ptr);

1573 /* only read priority level if frame is asynchronous */
1574 if (*svc_class_ptr == FDDI_SVC_ASYNC)
1575 op_pk_nfd_get (pkptr, "pri", pri_level_ptr);

1576 /* replace the packet on the proper subqueue */
1577 op_subq_pk_insert (i, pkptr, OPC_QPOS_HEAD);

1578 FOUT
1579 }

```

# APPENDIX C

## CPNI SINK "C" CODE

### "cp\_fddi\_sink.pr.c"

The line numbering in this appendix is within this thesis only, and does not correspond with that seen in OPNET's text editors.

```

1  /* Process model C form file: cp_fddi_sink.pr.c */
2  /* Portions of this file Copyright (C) MIL 3, Inc. 1992 */

3
4  /* OPNET system definitions */
5  #include <opnet.h>
6  #include "cp_fddi_sink.pr.h"
7  FSM_EXT_DECS

8
9  /* Header block */
10 /* Globals */
11 /* array format installed 20JAN94; positions 0-7 represent the asynch priority levels, PRIORI
12 /* represents synch traffic, and grand totals are as given in the original.

13 #define PRIORITIES 8 /* 20JAN94 */
14 #define XMITTER_ONE 0 /*10MAY94*/
15 #define XMITTER_TWO 1
16 #define XMITTER_THREE 2
17 #define XMITTER_FOUR 3

18
19 static /* 05FEB94 */
20 double fddi_sink_accum_delay = 0.0;
21 static /* 05FEB94 */
22 double fddi_sink_accum_delay_a[PRIORITIES+1]={0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
23 static /* 05FEB94 */
24 int fddi_sink_total_pkts = 0;
25 static /* 05FEB94 */
26 int fddi_sink_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
27 static /* 05FEB94 */
28 double fddi_sink_total_bits = 0.0;
29 static /* 05FEB94 */
30 double fddi_sink_total_bits_a[PRIORITIES+1]={0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
31 static /* 05FEB94 */
32 double fddi_sink_peak_delay = 0.0;
33 static /* 05FEB94 */
34 double fddi_sink_peak_delay_a[PRIORITIES+2]={0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

```

```

30 static /* 05FEB94 */
31 int fddi_sink_scalar_write = 0;
32 static /* 05FEB94 */
33 int pri_set = 20; /* 20JAN94 */
34 static
35 int subq_index = 0; /* 5APR94 */
36 static
37 int prev_src_addr[4]={0, 1, 2, 3}; /*25APR94*/
38 double buffer[4]={0.0,0.0,0.0,0.0}; /*10MAY94*/

39 /* statistics used for CDL throughput */
40 static /* 20APR94 */
41 int fddilpi_total_pkts = 0;
42 static
43 int fddilpi_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
44 static
45 double fddilpi_total_bits = 0.0;
46 static
47 double fddilpi_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

48/* Externally defined globals. */
49 extern double fddi_t_opr [];

50 /*12JAN94:attributes from the Environment file */
51 double Offered_Load; /* 12JAN94 */
52 double Asynch_Offered_Load; /* 12JAN94 */

53 /* transition expressions */
54 #define END_OF_SIM op_intrpt_type() == OPC_INTRPT_ENDSIM

55 /* State variable definitions */
56 typedef struct
57 {
58 FSM_SYS_STATE
59 Gshandle sv_thru_gshandle;
60 Gshandle sv_m_delay_gshandle;
61 Gshandle sv_ete_delay_gshandle;
62 Gshandle sv_thru_gshandle_a[10];
63 Gshandle sv_m_delay_gshandle_a[10];
64 Gshandle sv_ete_delay_gshandle_a[9];
65 Gshandle sv_t_gshandle;
66 Gshandle sv_t_gshandle_a[10];
67 Objid sv_my_id;
68 } cp_fddi_sink_state;

69 #define pr_state_ptr ((cp_fddi_sink_state*) SimI_Mod_State_Ptr)

```

```

70 #define thru_gshandle      pr_state_ptr->sv_thru_gshandle
71 #define m_delay_gshandle   pr_state_ptr->sv_m_delay_gshandle
72 #define ete_delay_gshandle pr_state_ptr->sv_ete_delay_gshandle
73 #define thru_gshandle_a    pr_state_ptr->sv_thru_gshandle_a
74 #define m_delay_gshandle_a pr_state_ptr->sv_m_delay_gshandle_a
75 #define ete_delay_gshandle_a pr_state_ptr->sv_ete_delay_gshandle_a
76 #define t_gshandle         pr_state_ptr->sv_t_gshandle
77 #define t_gshandle_a       pr_state_ptr->sv_t_gshandle_a
78 #define my_id               pr_state_ptr->sv_my_id

```

```

79 /* Process model interrupt handling procedure */

```

```

80 void
81 cp_fddi_sink ()
82 {
83     double delay, creat_time;
84     Packet* pkptr;
85     Packet*      pkptr1 ; /*5APR94*/
86     int src_addr, my_addr;
87     int dest_addr; /*14APR94*/
88     Ici* from_mac_ici_ptr;
89     double fddi_sink_ttrt;
90     int  xmit_subq_index; /*5APR94*/
91     int load_balance_code; /*6APR94*/
92     int i, subq_no; /*25APR94*/
93     int index; /*10MAY94 */

94     FSM_ENTER (cp_fddi_sink)

95     FSM_BLOCK_SWITCH
96     {
97         /*-----*/
98         /** state (DISCARD) enter executives **/
99         FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "DISCARD")
100     {
101         /* determine type of interrupt:10MAY94 */
102         switch (op_intrpt_type())
103         {
104             case OPC_INTRPT_STAT:
105                 /* the interrupt is caused by the transmitters' status */
106                 {
107                     index = op_intrpt_stat();
108                     switch(index)
109                     {

```

```

109 case XMITTER_ONE:
110 {
111 buffer[0] = op_stat_local_read(XMITTER_ONE);
112 break;
113 }
114 case XMITTER_TWO:
115 {
116 buffer[1] = op_stat_local_read(XMITTER_TWO);
117 break;
118 }
119 case XMITTER_THREE:
120 {
121 buffer[2] = op_stat_local_read(XMITTER_THREE);
122 break;
123 }
124 case XMITTER_FOUR:
125 {
126 buffer[3] = op_stat_local_read(XMITTER_FOUR);
127 break;
128 }
129 default:
130 {
131 op_sim_end("*** FDDI-CDL : FATAL ERROR","Unexpected stat interrupt","", "");
132 }
133 }
134 break;
135 }
136 case OPC_INTRPT_STRM:
137 /* the interrupt is caused by the incoming packets */
138 {
139 /* get the packet and the interface control info */
140 pkptr = op_pk_get (op_intrpt_strm ());
141 from_mac_ici_ptr = op_intrpt_ici ();

142 /* 20JAN94: get the packet's priority level, which */
143 /* will be used to index arrays of thruput and delay */
144 /* computations. */
145 /* pri_set = op_pk_priority_get (pkptr); doesn't work here */
146 op_pk_nfd_get (pkptr, "pri", &pri_set); /* 29JAN94 */

147 /* determine the time of creation of the packet */
148 op_pk_nfd_get (pkptr, "cr_time", &creat_time);

149 /* 18APR94:determine the destination address of the packet */
150 op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

151 /* 20APR94:determine the source address of the packet */
152 op_pk_nfd_get (pkptr, "src_addr", &src_addr);

153 /* 7APR94:determine id of own processor to use in finding */
154 /* load balancing attribute and station address of the bridge node */

```



```

155 my_id = op_id_self();

156 /* 7APR94: determine which load balancing algorithm is in use */
157 op_ima_obj_attr_get ( my_id, "load balancing algorithm", &load_balance_code );

158 /* 14APR94 : also get my own address */
159 op_ima_obj_attr_get ( my_id, "station_address", &my_addr);

160 /* destroy the packet */
161 /* op_pk_destroy (pkptr); */
162 /* 03FEB94: rather, enqueue the packet. This will be the */
163 /* first step toward developing a LAN bridging structure. */
164 /* -Nix */
165 /* op_subq_pk_insert (pri_set, pkptr, OPC_QPOS_TAIL); */

166 /* 14APR94: check the frame passed to "llc" is destined for */
167 /* this station. If it is destroy the packet and update the local traffic */
168 /* statistics; if not, allocate the packets */
169 /* to the transmitters since they are destined for the remote lan */
170 /* update also incoming return link statistics for the frames */
171 /* which will be queued in llc_sink to be sent to remote lan.*/
172 /* -Karayakaylar */

173 if((dest_addr == my_addr)&&(src_addr < my_addr))
174 {
175 /* add in its size */
176 fddi_sink_total_bits += op_pk_total_size_get (pkptr);
177 fddi_sink_total_bits_a[pri_set] += op_pk_total_size_get (pkptr); /* 20JAN-20APR94 */

178 /* accumulate delays */
179 delay = op_sim_time () - creat_time;
180 fddi_sink_accum_delay += delay;
181 fddi_sink_accum_delay_a[pri_set] += delay; /* 20JAN-20APR94 */

182 /* keep track of peak delay value */
183 if (delay > fddi_sink_peak_delay)
184 fddi_sink_peak_delay = delay;

185 /* 20JAN94: keep track by priority levels as well 23JAN-20APR94 */
186 if (delay > fddi_sink_peak_delay_a[pri_set])
187     fddi_sink_peak_delay_a[pri_set] = delay;

188 op_pk_destroy (pkptr);

189 /* increment packet counter; 20JAN94 */
190 fddi_sink_total_pkts++;
191 fddi_sink_total_pkts_a[pri_set]++;

192 /* if a multiple of 25 packets is reached, update stats */
193 /* 03FEB94: [0]->[7] represent asynch priorities 1->8, */
194 /* respectively; [8] represents synchronous traffic, */

```

```

195 /* and [9] represents overall asynchronous traffic.-Nix */
196 if (fddi_sink_total_pkts % 25 == 0)
197 {
198   op_stat_global_write (thru_gshandle,
199   fddi_sink_total_bits / op_sim_time ());

200   op_stat_global_write (thru_gshandle_a[pri_set],
201   fddi_sink_total_bits_a[0] / op_sim_time());
202   op_stat_global_write (thru_gshandle_a[0],
203   fddi_sink_total_bits_a[1] / op_sim_time());
204   op_stat_global_write (thru_gshandle_a[1],
205   fddi_sink_total_bits_a[pri_set] / op_sim_time());
206   op_stat_global_write (thru_gshandle_a[2],
207   fddi_sink_total_bits_a[2] / op_sim_time());
208   op_stat_global_write (thru_gshandle_a[3],
209   fddi_sink_total_bits_a[3] / op_sim_time());
210   op_stat_global_write (thru_gshandle_a[4],
211   fddi_sink_total_bits_a[4] / op_sim_time());
212   op_stat_global_write (thru_gshandle_a[5],
213   fddi_sink_total_bits_a[5] / op_sim_time());
214   op_stat_global_write (thru_gshandle_a[6],
215   fddi_sink_total_bits_a[6] / op_sim_time());
216   op_stat_global_write (thru_gshandle_a[7],
217   fddi_sink_total_bits_a[7] / op_sim_time());
218   op_stat_global_write (thru_gshandle_a[8],
219   fddi_sink_total_bits_a[8] / op_sim_time());

220 /* 30JAN94: gather all asynch stats into one overall figure */
221   op_stat_global_write (thru_gshandle_a[9],
222   (fddi_sink_total_bits - fddi_sink_total_bits_a[8]) /
223   op_sim_time());

224   /* (fddi_sink_total_bits_a[0] + fddi_sink_total_bits_a[1] + */
225   /* fddi_sink_total_bits_a[2] + fddi_sink_total_bits_a[3] + */
226   /* fddi_sink_total_bits_a[4] + fddi_sink_total_bits_a[5] + */
227   /* fddi_sink_total_bits_a[6] + fddi_sink_total_bits_a[7]) / */
228   /* op_sim_time()); */

229   op_stat_global_write (m_delay_gshandle,
230   fddi_sink_accum_delay / fddi_sink_total_pkts);

231   op_stat_global_write (m_delay_gshandle_a[0],
232   fddi_sink_accum_delay_a[0] / fddi_sink_total_pkts_a[0]);
233   op_stat_global_write (m_delay_gshandle_a[1],
234   fddi_sink_accum_delay_a[1] / fddi_sink_total_pkts_a[1]);
235   op_stat_global_write (m_delay_gshandle_a[2],
236   fddi_sink_accum_delay_a[2] / fddi_sink_total_pkts_a[2]);
237   op_stat_global_write (m_delay_gshandle_a[3],
238   fddi_sink_accum_delay_a[3] / fddi_sink_total_pkts_a[3]);

```

```

239     op_stat_global_write (m_delay_gshandle_a[4],
240                           fddi_sink_accum_delay_a[4] / fddi_sink_total_pkts_a[4]);
241     op_stat_global_write (m_delay_gshandle_a[5],
242                           fddi_sink_accum_delay_a[5] / fddi_sink_total_pkts_a[5]);
243     op_stat_global_write (m_delay_gshandle_a[6],
244                           fddi_sink_accum_delay_a[6] / fddi_sink_total_pkts_a[6]);
245     op_stat_global_write (m_delay_gshandle_a[7],
246                           fddi_sink_accum_delay_a[7] / fddi_sink_total_pkts_a[7]);
247     op_stat_global_write (m_delay_gshandle_a[8],
248                           fddi_sink_accum_delay_a[8] / fddi_sink_total_pkts_a[8]);

249 /* 30JAN94: gather all asynch stats into one figure */
250     op_stat_global_write (m_delay_gshandle_a[9],
251                           (fddi_sink_accum_delay - fddi_sink_accum_delay_a[8]) /
252                           (fddi_sink_total_pkts - fddi_sink_total_pkts_a[8]));

253     /*      (fddi_sink_accum_delay_a[0] + fddi_sink_accum_delay_a[1] +      */
254     /*      fddi_sink_accum_delay_a[2] + fddi_sink_accum_delay_a[3] +      */
255     /*      fddi_sink_accum_delay_a[4] + fddi_sink_accum_delay_a[5] +      */
256     /*      fddi_sink_accum_delay_a[6] + fddi_sink_accum_delay_a[7]) /      */
257     /*      (fddi_sink_total_pkts_a[0] + fddi_sink_total_pkts_a[1] +      */
258     /*      fddi_sink_total_pkts_a[2] + fddi_sink_total_pkts_a[3] +      */
259     /*      fddi_sink_total_pkts_a[4] + fddi_sink_total_pkts_a[5] +      */
260     /*      fddi_sink_total_pkts_a[6] + fddi_sink_total_pkts_a[7]));      */

261 /* also record actual delay values */
262 op_stat_global_write (ete_delay_gshandle, delay);
263 op_stat_global_write (ete_delay_gshandle_a[pri_set], delay);
264 }
265 } /* end of if(dest_addr == my_addr)&&(src_addr < my_addr) statement */

266 /* 20APR94:destroy the packets coming from the remote lan destined for this*/
267 /* station. These packets are not counted for local traffic.*/
268 else if(dest_addr == my_addr)
269 op_pk_destroy(pkptr);

270 /* 20APR94: check the frame passed to "llc" is destined for remote lan */
271 /* This will allow only the packets to be counted for CDL traffic.*/
272 /* -Karayakaylar */
273 else
274 {

275 /* add in its size */
276 fddilpi_total_bits += op_pk_total_size_get (pkptr);
277 fddilpi_total_bits_a[pri_set] += op_pk_total_size_get (pkptr); /* 20APR94 */

278 /* increment packet counter; 20APR94 */
279 fddilpi_total_pkts++;

```

```

280 fddilp1_total_pkts_a[pri_set]++;

281 /* if a multiple of 25 packets is reached, update stats */
282 /* [0]->[7] represent asynch priorities 1->8, */
283 /* respectively; [8] represents synchronous traffic, */
284 /* and [9] represents overall asynchronous traffic.-Nix */
285 if (fddilp1_total_pkts % 25 == 0)
286 {
287   op_stat_global_write (t_gshandle,
288     fddilp1_total_bits / op_sim_time ());

289     op_stat_global_write (t_gshandle_a[pri_set],
290       fddilp1_total_bits_a[0] / op_sim_time());
291     op_stat_global_write (t_gshandle_a[0],
292       fddilp1_total_bits_a[1] / op_sim_time());
293     op_stat_global_write (t_gshandle_a[1],
294       fddilp1_total_bits_a[pri_set] / op_sim_time());
295     op_stat_global_write (t_gshandle_a[2],
296       fddilp1_total_bits_a[2] / op_sim_time());
297     op_stat_global_write (t_gshandle_a[3],
298       fddilp1_total_bits_a[3] / op_sim_time());
299     op_stat_global_write (t_gshandle_a[4],
300       fddilp1_total_bits_a[4] / op_sim_time());
301     op_stat_global_write (t_gshandle_a[5],
302       fddilp1_total_bits_a[5] / op_sim_time());
303     op_stat_global_write (t_gshandle_a[6],
304       fddilp1_total_bits_a[6] / op_sim_time());
305     op_stat_global_write (t_gshandle_a[7],
306       fddilp1_total_bits_a[7] / op_sim_time());
307     op_stat_global_write (t_gshandle_a[8],
308       fddilp1_total_bits_a[8] / op_sim_time());

309 /* gather all asynch stats into one overall figure */
310   op_stat_global_write (t_gshandle_a[9],
311     (fddilp1_total_bits - fddilp1_total_bits_a[8]) /
312     op_sim_time());

313     /* (fddilp1_total_bits_a[0] + fddilp1_total_bits_a[1] + */
314     /* fddilp1_total_bits_a[2] + fddilp1_total_bits_a[3] + */
315     /* fddilp1_total_bits_a[4] + fddilp1_total_bits_a[5] + */
316     /* fddilp1_total_bits_a[6] + fddilp1_total_bits_a[7]) / */
317     /* op_sim_time()); */

318   }

319 /* 14APR94 :allocate the packets to llc_sink subqueues */
320 /* 6APR94 -Karayakaylar*/

```

```

321 /* check if load balancing algorithm is circular */
322 /* zero(0) is the circular load balancing code */
323 if (load_balance_code == 0)
324 {
325     /* 5APR94 */
326 /* Apply load balancing to insert the packets in the */
327 /* subqueues, in a circular order */
328 i = subq_index % 4;
329 /* check if previous source address is allocated to this queue */
330 /* if so, allocate the packet to that subqueue so that consecutive packets */
331 /* coming from the same station follow the same channel */
332 /* otherwise, allocate the packet to the next queue for transmission */
333 if(prev_src_addr[i] == src_addr)
334 {
335 op_subq_pk_insert(i, pkptr, OPC_QPOS_TAIL);
336 prev_src_addr[i] = src_addr;
337 }
338 else
339 {
340     subq_index++;
341 subq_no = subq_index % 4;
342 op_subq_pk_insert(subq_no, pkptr, OPC_QPOS_TAIL);
343 prev_src_addr[subq_no] = src_addr;
344 }
345     }

346     /* 25APR94 */
347 /* check if load balancing algorithm is empty allocation */
348 /* one(1) is the empty allocation load balancing code */
349 if (load_balance_code == 1)
350 {
351 i = subq_index % 4;
352 /* check if previous source address is allocated to this queue */
353     if(prev_src_addr[i] == src_addr)
354 {
355 op_subq_pk_insert(i, pkptr, OPC_QPOS_TAIL);
356 prev_src_addr[i] = src_addr;
357 }
358 else
359 /* Apply load balancing to insert the packets in the */
360 /* subqueues by choosing the subqueue which has the maximum current */
361 /* number of free packet slots */
362 {
363 subq_no = op_subq_index_map(OPC_QSEL_MAX_FREE_PKSIZE);
364 op_subq_pk_insert(subq_no, pkptr, OPC_QPOS_TAIL);
365 prev_src_addr[subq_no] = src_addr;
366 subq_index++;
367 }
368     }

```

```

369 /* send the packets to the transmitters */
370 xmit_subq_index = i;
371 /* check if this subqueue is empty and transmitter is not busy */
372 if ((!op_subq_empty(xmit_subq_index)) && (buffer[xmit_subq_index] == 0.0))
373 {
374 /*access the first packet in the subqueue */
375 pkptr1 = op_subq_pk_remove (xmit_subq_index, OPC_QPOS_HEAD);
376 /* forward it to the destination xmitter */
377 /* associated with the subqueue index */
378 op_pk_send (pkptr1, xmit_subq_index );
379 }
380 /*if(dest_addr > my_addr) statement */
381 break;
382 /* end of case OPC_INTRPT_STRM statement */

383 /* end of switch */
384 }

385 /** blocking after enter executives of unforced state. **/
386 FSM_EXIT (1, cp_fddi_sink)

387 /** state (DISCARD) exit executives **/
388 FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "DISCARD")
389 {
390 }

391 /** state (DISCARD) transition processing **/
392 FSM_INIT_COND (END_OF_SIM)
393 FSM_DFLT_COND
394 FSM_TEST_LOGIC ("DISCARD")

395 FSM_TRANSIT_SWITCH
396 {
397 FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
398 FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
399 }
400 /*-----*/

401 /** state (STATS) enter executives **/
402 FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "STATS")
403 {
404 /* At end of simulation, scalar performance statistics */
405 /* and input parameters are written out. */

406 op_stat_scalar_write ("RL Throughput (bps), Priority 1",
407 fddilpi_total_bits_a[0] / op_sim_time ()); /*20APR94*/

```

```

408 op_stat_scalar_write ("RL Throughput (bps), Priority 2",
409     fddilpi_total_bits_a[1] / op_sim_time ());

410 op_stat_scalar_write ("RL Throughput (bps), Priority 3",
411     fddilpi_total_bits_a[2] / op_sim_time ());

412 op_stat_scalar_write ("RL Throughput (bps), Priority 4",
413     fddilpi_total_bits_a[3] / op_sim_time ());

414 op_stat_scalar_write ("RL Throughput (bps), Priority 5",
415     fddilpi_total_bits_a[4] / op_sim_time ());

416 op_stat_scalar_write ("RL Throughput (bps), Priority 6",
417     fddilpi_total_bits_a[5] / op_sim_time ());

418 op_stat_scalar_write ("RL Throughput (bps), Priority 7",
419     fddilpi_total_bits_a[6] / op_sim_time ());

420 op_stat_scalar_write ("RL Throughput (bps), Priority 8",
421     fddilpi_total_bits_a[7] / op_sim_time ());

422 op_stat_scalar_write ("RL Throughput (bps), Asynchronous",
423     (fddilpi_total_bits - fddilpi_total_bits_a[8]) / op_sim_time ());

424 /*      (fddilpi_total_bits_a[0] + fddilpi_total_bits_a[1] + */
425 /*      fddilpi_total_bits_a[2] + fddilpi_total_bits_a[3] + */
426 /*      fddilpi_total_bits_a[4] + fddilpi_total_bits_a[5] + */
427 /*      fddilpi_total_bits_a[6] + fddilpi_total_bits_a[7]) / */
428 /*      op_sim_time ()); */

429 op_stat_scalar_write ("RL Throughput (bps), Synchronous",
430     fddilpi_total_bits_a[8] / op_sim_time ());

431 op_stat_scalar_write ("RL Throughput (bps), Total",
432     fddilpi_total_bits / op_sim_time ()); /*20APR94*/

433 /* Only one station needs to do this */
434 if (!fddi_sink_scalar_write)
435 {
436 /* set the scalar write flag */
437 fddi_sink_scalar_write = 1;

438 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 1",
439 fddi_sink_accum_delay_a[0] / fddi_sink_total_pkts_a[0]);

```

```

440 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 2",
441     fddi_sink_accum_delay_a[1] / fddi_sink_total_pkts_a[1]);

442 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 3",
443     fddi_sink_accum_delay_a[2] / fddi_sink_total_pkts_a[2]);

444 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 4",
445     fddi_sink_accum_delay_a[3] / fddi_sink_total_pkts_a[3]);

446 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 5",
447     fddi_sink_accum_delay_a[4] / fddi_sink_total_pkts_a[4]);

448 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 6",
449     fddi_sink_accum_delay_a[5] / fddi_sink_total_pkts_a[5]);

450 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 7",
451     fddi_sink_accum_delay_a[6] / fddi_sink_total_pkts_a[6]);

452 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 8",
453     fddi_sink_accum_delay_a[7] / fddi_sink_total_pkts_a[7]);

454 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Asynchronous",
455     (fddi_sink_accum_delay - fddi_sink_accum_delay_a[8]) /
456     (fddi_sink_total_pkts - fddi_sink_total_pkts_a[8]));

457 /* (fddi_sink_accum_delay_a[0] + fddi_sink_accum_delay_a[1] + */
458 /* fddi_sink_accum_delay_a[2] + fddi_sink_accum_delay_a[3] + */
459 /* fddi_sink_accum_delay_a[4] + fddi_sink_accum_delay_a[5] + */
460 /* fddi_sink_accum_delay_a[6] + fddi_sink_accum_delay_a[7]) / */
461 /* (fddi_sink_total_pkts_a[0] + fddi_sink_total_pkts_a[1] + */
462 /* fddi_sink_total_pkts_a[2] + fddi_sink_total_pkts_a[3] + */
463 /* fddi_sink_total_pkts_a[4] + fddi_sink_total_pkts_a[5] + */
464 /* fddi_sink_total_pkts_a[6] + fddi_sink_total_pkts_a[7])); */

465 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Synchronous",
466     fddi_sink_accum_delay_a[8] / fddi_sink_total_pkts_a[8]);

467 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Total",
468     fddi_sink_accum_delay / fddi_sink_total_pkts);

469 op_stat_scalar_write ("Throughput-0 (bps), Priority 1",
470     fddi_sink_total_bits_a[0] / op_sim_time ());

471 op_stat_scalar_write ("Throughput-0 (bps), Priority 2",
472     fddi_sink_total_bits_a[1] / op_sim_time ());

473 op_stat_scalar_write ("Throughput-0 (bps), Priority 3",
474     fddi_sink_total_bits_a[2] / op_sim_time ());

475 op_stat_scalar_write ("Throughput-0 (bps), Priority 4",

```



```

476         fddi_sink_total_bits_a[3] / op_sim_time ());

477     op_stat_scalar_write ("Throughput-0 (bps), Priority 5",
478         fddi_sink_total_bits_a[4] / op_sim_time ());

479     op_stat_scalar_write ("Throughput-0 (bps), Priority 6",
480         fddi_sink_total_bits_a[5] / op_sim_time ());

481     op_stat_scalar_write ("Throughput-0 (bps), Priority 7",
482         fddi_sink_total_bits_a[6] / op_sim_time ());

483     op_stat_scalar_write ("Throughput-0 (bps), Priority 8",
484         fddi_sink_total_bits_a[7] / op_sim_time ());

485     op_stat_scalar_write ("Throughput-0 (bps), Asynchronous",
486         (fddi_sink_total_bits - fddi_sink_total_bits_a[8]) / op_sim_time ());

487     /*      (fddi_sink_total_bits_a[0] + fddi_sink_total_bits_a[1] + */
488     /*      fddi_sink_total_bits_a[2] + fddi_sink_total_bits_a[3] + */
489     /*      fddi_sink_total_bits_a[4] + fddi_sink_total_bits_a[5] + */
490     /*      fddi_sink_total_bits_a[6] + fddi_sink_total_bits_a[7]) / */
491     /*      op_sim_time ()); */

492     op_stat_scalar_write ("Throughput-0 (bps), Synchronous",
493         fddi_sink_total_bits_a[8] / op_sim_time ());

494     op_stat_scalar_write ("Throughput-0 (bps), Total",
495         fddi_sink_total_bits / op_sim_time ());

496 op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 1",
497 fddi_sink_peak_delay_a[0]);

498     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 2",
499         fddi_sink_peak_delay_a[1]);
500     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 3",
501         fddi_sink_peak_delay_a[2]);

501     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 4",
502         fddi_sink_peak_delay_a[3]);

503     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 5",
504         fddi_sink_peak_delay_a[4]);

505     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 6",
506         fddi_sink_peak_delay_a[5]);

507     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 7",

```

```

508         fddi_sink_peak_delay_a[6]);

509     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 8",
510         fddi_sink_peak_delay_a[7]);

511     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Synchronous",
512         fddi_sink_peak_delay_a[8]);

513     op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Overall",
514         fddi_sink_peak_delay);

515 /* Write the TTRT value for ring 0. This preserves */
516 /* the old behavior for single-ring simulations. */
517 op_stat_scalar_write ("TTRT (sec.) - Ring 0",
518     fddi_t_opr [0]);

519 /* 12JAN94: obtain offered load information from the Environment */
520 /* file; this will be used to provide abscissa information that */
521 /* can be plotted in the Analysis Editor (see "fddi_sink" STATS */
522 /* state. To the user: it's your job to keep these current in */
523 /* the Environment File. -Nix */
524 op_ima_sim_attr_get (OPC_IMA_DOUBLE, "total_offered_load_0", &Offered_Load);
525 op_ima_sim_attr_get (OPC_IMA_DOUBLE, "asynch_offered_load_0", &Asynch_Offered_Load);

526 /* 12JAN94: write the total offered load for this run */
527 op_stat_scalar_write ("Total Offered Load-0 (Mbps)",
528     Offered_Load);

529     op_stat_scalar_write ("Asynchronous Offered Load-0 (Mbps)",
530         Asynch_Offered_Load);
531 }
532 }

533 /** blocking after enter executives of unforced state. **/
534 FSM_EXIT (3,cp_fddi_sink)

535 /** state (STATS) exit executives **/
536 FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "STATS")
537 {
538 }

539 /** state (STATS) transition processing **/
540 FSM_TRANSIT_MISSING ("STATS")
541 /*-----*/

```

```

542 /** state (INIT) enter executives **/
543 FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "INIT")
544 {
545 /* get the gshandles of the global statistic to be obtained */
546 /* 20JAN94: set array format */

547 thru_gshandle_a[0] = op_stat_global_reg ("pri 1 throughput-0 (bps)");
548 thru_gshandle_a[1] = op_stat_global_reg ("pri 2 throughput-0 (bps)");
549 thru_gshandle_a[2] = op_stat_global_reg ("pri 3 throughput-0 (bps)");
550 thru_gshandle_a[3] = op_stat_global_reg ("pri 4 throughput-0 (bps)");
551 thru_gshandle_a[4] = op_stat_global_reg ("pri 5 throughput-0 (bps)");
552 thru_gshandle_a[5] = op_stat_global_reg ("pri 6 throughput-0 (bps)");
553 thru_gshandle_a[6] = op_stat_global_reg ("pri 7 throughput-0 (bps)");
554 thru_gshandle_a[7] = op_stat_global_reg ("pri 8 throughput-0 (bps)");
555 thru_gshandle_a[8] = op_stat_global_reg ("synch throughput-0 (bps)");
556 thru_gshandle_a[9] = op_stat_global_reg ("async throughput-0 (bps)");
557 thru_gshandle      = op_stat_global_reg ("total throughput-0 (bps)");

558 m_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 mean delay-0 (sec.)");
559 m_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 mean delay-0 (sec.)");
560 m_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 mean delay-0 (sec.)");
561 m_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 mean delay-0 (sec.)");
562 m_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 mean delay-0 (sec.)");
563 m_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 mean delay-0 (sec.)");
564 m_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 mean delay-0 (sec.)");
565 m_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 mean delay-0 (sec.)");
566 m_delay_gshandle_a[8] = op_stat_global_reg ("synch mean delay-0 (sec.)");
567 m_delay_gshandle_a[9] = op_stat_global_reg ("async mean delay-0 (sec.)");
568 m_delay_gshandle      = op_stat_global_reg ("total mean delay-0 (sec.)");

569 ete_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 end-to-end delay-0 (sec.)");
570 ete_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 end-to-end delay-0 (sec.)");
571 ete_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 end-to-end delay-0 (sec.)");
572 ete_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 end-to-end delay-0 (sec.)");
573 ete_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 end-to-end delay-0 (sec.)");
574 ete_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 end-to-end delay-0 (sec.)");
575 ete_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 end-to-end delay-0 (sec.)");
576 ete_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 end-to-end delay-0 (sec.)");
577 ete_delay_gshandle_a[8] = op_stat_global_reg ("synch end-to-end delay-0 (sec.)");
578 ete_delay_gshandle      = op_stat_global_reg ("total end-to-end delay-0 (sec.)");

579 t_gshandle_a[0] = op_stat_global_reg ("pri 1 RL throughput (bps)"); /*20APR94*/
580 t_gshandle_a[1] = op_stat_global_reg ("pri 2 RL throughput (bps)");
581 t_gshandle_a[2] = op_stat_global_reg ("pri 3 RL throughput (bps)");
582 t_gshandle_a[3] = op_stat_global_reg ("pri 4 RL throughput (bps)");
583 t_gshandle_a[4] = op_stat_global_reg ("pri 5 RL throughput (bps)");
584 t_gshandle_a[5] = op_stat_global_reg ("pri 6 RL throughput (bps)");
585 t_gshandle_a[6] = op_stat_global_reg ("pri 7 RL throughput (bps)");
586 t_gshandle_a[7] = op_stat_global_reg ("pri 8 RL throughput (bps)");
587 t_gshandle_a[8] = op_stat_global_reg ("synch RL throughput (bps)");

```

```

588 t_ghandle_a[9] = op_stat_global_reg ("async RL throughput (bps)");
589 t_ghandle      = op_stat_global_reg ("total RL throughput (bps)");

590 }

591 /** state (INIT) exit executives **/
592 FSM_STATE_EXIT_FORCED (2, state2_exit_exec, "INIT")
593 {
594 }

595 /** state (INIT) transition processing **/
596 FSM_INIT_COND (END_OF_SIM)
597 FSM_DFLT_COND
598 FSM_TEST_LOGIC ("INIT")

599 FSM_TRANSIT_SWITCH
600 {
601 FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
602 FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
603 }
604 /*-----*/

605 }

606 FSM_EXIT (2, cp_fddi_sink)
607 }

608 void
609 cp_fddi_sink_svar (prs_ptr, var_name, var_p_ptr)
610 cp_fddi_sink_state *prs_ptr;
611 char *var_name, **var_p_ptr;
612 {

613 FIN (cp_fddi_sink_svar (prs_ptr))

614 *var_p_ptr = VOS_NIL;
615 if (Vos_String_Equal ("thru_gshandle" , var_name))
616 *var_p_ptr = (char *) (&prs_ptr->sv_thru_gshandle);
617 if (Vos_String_Equal ("m_delay_gshandle" , var_name))
618 *var_p_ptr = (char *) (&prs_ptr->sv_m_delay_gshandle);
619 if (Vos_String_Equal ("ete_delay_gshandle" , var_name))

```

```

620 *var_p_ptr = (char *) (&prs_ptr->sv_ete_delay_gshandle);
621 if (Vos_String_Equal ("thru_gshandle_a" , var_name))
622 *var_p_ptr = (char *) (prs_ptr->sv_thru_gshandle_a);
623 if (Vos_String_Equal ("m_delay_gshandle_a" , var_name))
624 *var_p_ptr = (char *) (prs_ptr->sv_m_delay_gshandle_a);
625 if (Vos_String_Equal ("ete_delay_gshandle_a" , var_name))
626 *var_p_ptr = (char *) (prs_ptr->sv_ete_delay_gshandle_a);
627 if (Vos_String_Equal ("t_gshandle" , var_name))
628 *var_p_ptr = (char *) (&prs_ptr->sv_t_gshandle);
629 if (Vos_String_Equal ("t_gshandle_a" , var_name))
630 *var_p_ptr = (char *) (prs_ptr->sv_t_gshandle_a);
631 if (Vos_String_Equal ("my_id" , var_name))
632 *var_p_ptr = (char *) (&prs_ptr->sv_my_id);

```

```

633 FOOT;
634 }

```

```

635 void
636 cp_fddi_sink_diag ()
637 {
638 double delay, creat_time;
639 Packet* pkptr;
640 Packet*      pkptr1 ; /*5APR94*/
641 int src_addr, my_addr;
642 int dest_addr; /*14APR94*/
643 Ici* from_mac_ici_ptr;
644 double fddi_sink_ttrt;
645 int xmit_subq_index; /*5APR94*/
646 int load_balance_code; /*6APR94*/
647 int i, subq_no; /*25APR94*/
648 int index; /*10MAY94 */

```

```

649 FIN (cp_fddi_sink_diag ())

```

```

650 FOOT;
651 }

```

```

652 void
653 cp_fddi_sink_terminate ()
654 {
655 double delay, creat_time;
656 Packet* pkptr;
657 Packet*      pkptr1 ; /*5APR94*/

```

```

658 int src_addr, my_addr;
659 int dest_addr; /*14APR94*/
660 Ici* from_mac_ici_ptr;
661 double fddi_sink_ttrt;
662 int xmit_subq_index; /*5APR94*/
663 int load_balance_code; /*6APR94*/
664 int i, subq_no; /*25APR94*/
665 int index; /*10MAY94 */

666 FIN (cp_fddi_sink_terminate ())

667 FOOT;
668 }

669 Compcode
670 cp_fddi_sink_init (pr_state_pptr)
671 cp_fddi_sink_state **pr_state_pptr;
672 {
673 static VosT_Cm_Obtype obtype = OPC_NIL;

674 FIN (cp_fddi_sink_init (pr_state_pptr))

675 if (obtype == OPC_NIL)
676 {
677 if (Vos_Catmem_Register ("proc state vars (cp_fddi_sink)",
678 sizeof (cp_fddi_sink_state), Vos_Nop, &obtype) == VOSC_FAILURE)
679 FRET (OPC_COMPCODE_FAILURE)
680 }

681 if ((*pr_state_pptr = (cp_fddi_sink_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
682 FRET (OPC_COMPCODE_FAILURE)
683 else
684 {
685 (*pr_state_pptr)->current_block = 4;
686 FRET (OPC_COMPCODE_SUCCESS)
687 }
688 }

```

## APPENDIX D

### SPNI SOURCE "C" CODE

"sp\_fddi\_gen.pr.c"

The line numbering in this appendix is within this thesis only, and does not correspond with that seen in OPNET's text editors.

```
1 /* Process model C form file: sp_fddi_gen.pr.c */
2 /* Portions of this file Copyright (C) MIL 3, Inc. 1992 */

3
4
5 /* OPNET system definitions */
6 #include <opnet.h>
7 #include "sp_fddi_gen.pr.h"
8 FSM_EXT_DECS

9
10 /* Header block */
11 #define MAC_LAYER_OUT_STREAM 0
12 #define LLC_SINK_OUT_STREAM 1 /*18APR94*/

13
14 /* define possible service classes for frames */
15 #define FDDI_SVC_ASYNC 0
16 #define FDDI_SVC_SYNC 1

17
18 /* define token classes */
19 #define FDDI_TK_NONRESTRICTED 0
20 #define FDDI_TK_RESTRICTED 1

21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

27 int          sv_station_addr;
28 int          sv_src_addr;
29 int          sv_low_pkt_priority;
30 int          sv_high_pkt_priority;
31 double       sv_arrival_rate;
32 double       sv_mean_pk_len;
33 double       sv_async_mix;
34 Ici*         sv_mac_iciptr;
35 Ici*         sv_mac_iciptri;
36 Ici*         sv_llc_ici_ptr;
37 Packet*      sv_pkptri;
38 } sp_fddi_gen_state;

39 #define pr_state_ptr      ((sp_fddi_gen_state*) SimI_Mod_State_Ptr)
40 #define inter_dist_ptr   pr_state_ptr->sv_inter_dist_ptr
41 #define len_dist_ptr     pr_state_ptr->sv_len_dist_ptr
42 #define dest_dist_ptr    pr_state_ptr->sv_dest_dist_ptr
43 #define pkt_priority_ptr pr_state_ptr->sv_pkt_priority_ptr
44 #define mac_objid        pr_state_ptr->sv_mac_objid
45 #define my_id            pr_state_ptr->sv_my_id
46 #define low_dest_addr    pr_state_ptr->sv_low_dest_addr
47 #define high_dest_addr   pr_state_ptr->sv_high_dest_addr
48 #define station_addr     pr_state_ptr->sv_station_addr
49 #define src_addr         pr_state_ptr->sv_src_addr
50 #define low_pkt_priority pr_state_ptr->sv_low_pkt_priority
51 #define high_pkt_priority pr_state_ptr->sv_high_pkt_priority
52 #define arrival_rate     pr_state_ptr->sv_arrival_rate
53 #define mean_pk_len      pr_state_ptr->sv_mean_pk_len
54 #define async_mix        pr_state_ptr->sv_async_mix
55 #define mac_iciptr       pr_state_ptr->sv_mac_iciptr
56 #define mac_iciptri      pr_state_ptr->sv_mac_iciptri
57 #define llc_ici_ptr      pr_state_ptr->sv_llc_ici_ptr
58 #define pkptri           pr_state_ptr->sv_pkptri

59 /* Process model interrupt handling procedure */

60 void
61 sp_fddi_gen ()
62 {
63 Packet *pkptri;
64 int pklen;
65 int dest_addr;
66 int i, restricted;
67 int          pkt_prio;

```



```

68 FSM_ENTER (sp_fddi_gen)

69 FSM_BLOCK_SWITCH
70 {
71 /*-----*/
72 /** state (INIT) enter executives **/
73 FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "INIT")
74 {
75 /* determine id of own processor to use in finding attrs */
76 my_id = op_id_self ();

77 /* determine address range for uniform desination assignment */
78 op_ima_obj_attr_get (my_id, "low dest address", &low_dest_addr);
79 op_ima_obj_attr_get (my_id, "high dest address", &high_dest_addr);

80 /* determine object id of connected 'mac' layer process */
81 mac_objid = op_topo_assoc (my_id, OPC_TOPO_ASSOC_OUT,
82 OPC_OBJMNTYPE_MODULE, MAC_LAYER_OUT_STREAM);

83 /* determine the address assigned to it */
84 /* which is also the address of this station */
85 op_ima_obj_attr_get (mac_objid, "station_address", &station_addr);

86 /* set up a distribution for generation of addresses */
87 dest_dist_ptr = op_dist_load ("uniform_int", low_dest_addr,
88                               high_dest_addr);

89 /* added 26DEC93 */
90 /* determine priority range for uniform traffic generation */
91 op_ima_obj_attr_get (my_id, "high pkt priority", &high_pkt_priority);
92 op_ima_obj_attr_get (my_id, "low pkt priority", &low_pkt_priority);

93 /* set up a distribution for generation of priorities */
94 pkt_priority_ptr = op_dist_load ("uniform_int", low_pkt_priority, high_pkt_priority);

95 /* above added 26DEC93 */

96 /* also determine the arrival rate for packet generation */
97 op_ima_obj_attr_get (my_id, "arrival_rate", &arrival_rate);

98 /* determine the mix of asynchronous and synchronous */
99 /* traffic. This is expressed as the proportion of */
100 /* asynchronous traffic. i.e a value of 1.0 indicates */
101 /* that all the produced traffic shall be asynchronous. */
102 op_ima_obj_attr_get (my_id, "async_mix", &async_mix);

103 /* set up a distribution for arrival generations */
104 if (arrival_rate != 0.0)

```

```

105 {
106 /* arrivals are exponentially distributed, with given mean */
107 inter_dist_ptr = op_dist_load ("constant", 1.0 / arrival_rate, 0.0);

108 /* determine the distribution for packet size */
109 op_ima_obj_attr_get (my_id, "mean pk length", &mean_pk_len);

110 /* set up corresponding distribution */
111 len_dist_ptr = op_dist_load ("constant", mean_pk_len, 0.0);

112 /* designate the time of first arrival */
113 fddi_gen_schedule ();

114 /* set up an interface control information (ICI) structure */
115 /* to communicate parameters to the mac layer process */
116 /* (it is more efficient to set one up now and keep it */
117 /* as a state variable than to allocate one on each packet xfer) */
118 mac_iciptr = op_ici_create ("fddi_mac_req");
119 }

120 }

121 /** blocking after enter executives of unforced state. **/
122 FSM_EXIT (1, sp_fddi_gen)

123 /** state (INIT) exit executives **/
124 FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "INIT")
125 {
126 }

127 /** state (INIT) transition processing **/
128 FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
129 /*-----*/

130 /** state (ARRIVAL) enter executives **/
131 FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "ARRIVAL")
132 {
133 /* This station should receive frames from the other lan as long as */
134 /* there are frames in the input streams addressed to this lan */
135 /* check if the interrupt type is stream interrupt */
136 if (op_intrpt_type() == OPC_INTRPT_STRM)
137 {
138 /* if it is, get the packet in the input stream causing interrupt */
139 pkptr1 = op_pk_get(op_intrpt_strm());
140 /* get the destination address of the frame : 16APR94 */
141 op_pk_nfd_get(pkptr1, "dest_addr", &dest_addr);

```

```

142 /* check if this frame is for the remote bridge station(bridge in surface lan) */
143 if(dest_addr == station_addr)
144 /* if it is, send the packet to llc_sink directly */
145 /* in order to prevent overhead of mac access */
146 op_pk_send(pkptr1, LLC_SINK_OUT_STREAM);/*19APR94*/
147 else
148 /* this packet is to send to mac */
149 {
150 /* determine the source address of the frame */
151 op_pk_nfd_get(pkptr1, "src_addr", &src_addr);
152 /* set up an ICI structure to communicate parameters to */
153 /* MAC layer process */
154 mac_iciptr1 = op_ici_create("fddi_mac_req");
155 /* place the original source address into the ICI *//* 16APR94 */
156 /* "fddi_mac_req" is modified so that it contains the original */
157 /* source address from the local lan(collection platform) */
158 op_ici_attr_set(mac_iciptr1, "src_addr", src_addr);
159 /* place the destination address into the ICI */ /*12APR94*/
160 op_ici_attr_set(mac_iciptr1, "dest_addr", dest_addr);
161 /* assign the service class and requested token class */
162 /* At this moment the frames coming from the remote lan are assumed */
163 /* to have the same priority as synchronous frames in order not to accumulate */
164 /* packets on the bridge station mac and instead to deliver their destinations */
165 /* as soon as possible */
166 op_pk_nfd_set(pkptr1, "pri", 8);
167 op_ici_attr_set(mac_iciptr1, "svc_class", FDDI_SVC_SYNC);
168 op_ici_attr_set(mac_iciptr1, "pri", 8);
169 op_ici_attr_set(mac_iciptr1, "tk_class", FDDI_TK_NONRESTRICTED);
170 /* send the packet coupled with the ICI */
171 op_ici_install(mac_iciptr1);
172 op_pk_send(pkptr1, MAC_LAYER_OUT_STREAM);
173     }
174 }
175 /* otherwise, generate the frame */
176 else
177 {
178 /* determine the length of the packet to be generated */
179 pklen = op_dist_outcome (len_dist_ptr);

180 /* determine the destination */
181 /* dont allow this station's address as a possible outcome */
182 gen_packet:
183 dest_addr = op_dist_outcome (dest_dist_ptr);
184 if (dest_addr != -1 && dest_addr == station_addr)
185 goto gen_packet;

186 /* 26DEC94 & 29JAN94: determine its priority */
187 pkt_prio = op_dist_outcome (pkt_priority_ptr);

188 /* create a packet to send to mac */
189 pkptr = op_pk_create_fmt ("fddi_llc_fr");

```

```

190 /* assign its overall size. */
192 op_pk_total_size_set (pkptr, pklen);

193 /* assign the time of creation */
194 op_pk_nfd_set (pkptr, "cr_time", op_sim_time ());

195 /* place the destination address into the ICI */
196 /* (the protocol_type field will default) */
197 op_ici_attr_set (mac_iciptr, "dest_addr", dest_addr);

198 /* place the source address into the ICI */ /* 17APR94 */
199 op_ici_attr_set (mac_iciptr, "src_addr", station_addr);

200 /* assign the priority, and requested token class */
201 /* also assign the service class; 29JAN94: the fddi_llc_fr */
202 /* format is modified to include a "pri" field. */
203 if (op_dist_uniform (1.0) <= async_mix)
204 {
205     op_pk_nfd_set (pkptr, "pri", pkt_prio); /* 29JAN94 */
206     op_ici_attr_set (mac_iciptr, "svc_class", FDDI_SVC_ASYNC);
207     op_ici_attr_set (mac_iciptr, "pri", pkt_prio); /* 29JAN94 */
208 }
209 else{
210     op_pk_nfd_set (pkptr, "pri", 8); /* 29JAN94 */
211     op_ici_attr_set (mac_iciptr, "svc_class", FDDI_SVC_SYNC);
212     op_ici_attr_set (mac_iciptr, "pri", 8); /* 29JAN94 */
213 }

214 /* Request only nonrestricted tokens after transmission */
215 op_ici_attr_set (mac_iciptr, "tk_class", FDDI_TK_NONRESTRICTED);
216
217 /* Having determined priority, assign it; 26DEC93 */
218 /* op_ici_attr_set (mac_iciptr, "pri", pkt_prio); */

219 /* send the packet coupled with the ICI */
220 op_ici_install (mac_iciptr);
221 /* check if destination address is in the local lan(collection platform)*/
222 if(dest_addr <= 9)
223 /* if it is, this packet is to send llc_sink directly */
224 op_pk_send (pkptr, LLC_SINK_OUT_STREAM); /*18APR94*/
225 else
226 /* if not, the packet is destined for remote lan (surface stations)*/
227 op_pk_send (pkptr, MAC_LAYER_OUT_STREAM);

228 /* schedule the next arrival */
229 fddi_gen_schedule ();
230 }
231 }

```

```

232 /** blocking after enter executives of unforced state. **/
233 FSM_EXIT (3,sp_fddi_gen)

234 /** state (ARRIVAL) exit executives **/
235 FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "ARRIVAL")
236 {
237 }

238 /** state (ARRIVAL) transition processing **/
239 FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
240 /*-----*/

241 }

242 FSM_EXIT (0,sp_fddi_gen)
243 }

244 void
245 sp_fddi_gen_svar (prs_ptr,var_name,var_p_ptr)
246 sp_fddi_gen_state *prs_ptr;
247 char *var_name, **var_p_ptr;
248 {

249 FIN (sp_fddi_gen_svar (prs_ptr))

250 *var_p_ptr = VOS_NIL;
251 if (Vos_String_Equal ("inter_dist_ptr" , var_name))
252 *var_p_ptr = (char *) (&prs_ptr->sv_inter_dist_ptr);
253 if (Vos_String_Equal ("len_dist_ptr" , var_name))
254 *var_p_ptr = (char *) (&prs_ptr->sv_len_dist_ptr);
255 if (Vos_String_Equal ("dest_dist_ptr" , var_name))
256 *var_p_ptr = (char *) (&prs_ptr->sv_dest_dist_ptr);
257 if (Vos_String_Equal ("pkt_priority_ptr" , var_name))
258 *var_p_ptr = (char *) (&prs_ptr->sv_pkt_priority_ptr);
259 if (Vos_String_Equal ("mac_objid" , var_name))
260 *var_p_ptr = (char *) (&prs_ptr->sv_mac_objid);
261 if (Vos_String_Equal ("my_id" , var_name))
262 *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
263 if (Vos_String_Equal ("low_dest_addr" , var_name))
264 *var_p_ptr = (char *) (&prs_ptr->sv_low_dest_addr);
265 if (Vos_String_Equal ("high_dest_addr" , var_name))
266 *var_p_ptr = (char *) (&prs_ptr->sv_high_dest_addr);
267 if (Vos_String_Equal ("station_addr" , var_name))

```

```

268 *var_p_ptr = (char *) (&prs_ptr->sv_station_addr);
269 if (Vos_String_Equal ("src_addr" , var_name))
270 *var_p_ptr = (char *) (&prs_ptr->sv_src_addr);
271 if (Vos_String_Equal ("low_pkt_priority" , var_name))
272 *var_p_ptr = (char *) (&prs_ptr->sv_low_pkt_priority);
273 if (Vos_String_Equal ("high_pkt_priority" , var_name))
274 *var_p_ptr = (char *) (&prs_ptr->sv_high_pkt_priority);
275 if (Vos_String_Equal ("arrival_rate" , var_name))
276 *var_p_ptr = (char *) (&prs_ptr->sv_arrival_rate);
277 if (Vos_String_Equal ("mean_pk_len" , var_name))
278 *var_p_ptr = (char *) (&prs_ptr->sv_mean_pk_len);
279 if (Vos_String_Equal ("async_mix" , var_name))
280 *var_p_ptr = (char *) (&prs_ptr->sv_async_mix);
281 if (Vos_String_Equal ("mac_iciptr" , var_name))
282 *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr);
283 if (Vos_String_Equal ("mac_iciptri" , var_name))
284 *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptri);
285 if (Vos_String_Equal ("llc_ici_ptr" , var_name))
286 *var_p_ptr = (char *) (&prs_ptr->sv_llc_ici_ptr);
287 if (Vos_String_Equal ("pkptri" , var_name))
288 *var_p_ptr = (char *) (&prs_ptr->sv_pkptri);

289 FOUT;
290 }

```

```

291 void
292 sp_fddi_gen_diag ()
293 {
294 Packet *pkptr;
295 int pklen;
296 int dest_addr;
297 int i, restricted;
298 int          pkt_prio;

299 FIN (sp_fddi_gen_diag ())

```

```

300 FOUT;
301 }

```

```

302 void
303 sp_fddi_gen_terminate ()
304 {
305 Packet *pkptr;
306 int pklen;

```

```

307 int dest_addr;
308 int i, restricted;
309 int      pkt_prio;

310 FIN (sp_fddi_gen_terminate ())

311 FOUT;
312 }

313 Compcode
314 sp_fddi_gen_init (pr_state_pptr)
315 sp_fddi_gen_state **pr_state_pptr;
316 {
317 static VosT_Cm_Obtype obtype = OPC_NIL;

318 FIN (sp_fddi_gen_init (pr_state_pptr))

319 if (obtype == OPC_NIL)
320 {
321 if (Vos_Catmem_Register ("proc state vars (sp_fddi_gen)",
322 sizeof (sp_fddi_gen_state), Vos_Nop, &obtype) == VOSC_FAILURE)
323 FRET (OPC_COMPCODE_FAILURE)
324 }

325 if ((*pr_state_pptr = (sp_fddi_gen_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
326 FRET (OPC_COMPCODE_FAILURE)
327 else
328 {
329 (*pr_state_pptr)->current_block = 0;
330 FRET (OPC_COMPCODE_SUCCESS)
331 }
332 }

333 /* static added 2DEC93, on advice from MIL3 */
334 static
335 fddi_gen_schedule ()
336 {
337 double inter_time;

338 /* obtain an interarrival period according to the */
339 /* prescribed distribution */
340 inter_time = op_dist_outcome (inter_dist_ptr);

341 /* schedule the arrival of next generated packet */
342 op_intrpt_schedule_self (op_sim_time () + inter_time, 0);
343 }

```





# APPENDIX E

## SPNI MAC "C" CODE EXCERPT

### "sp\_fddi\_mac.pr.c"

The line numbering in this appendix is within this thesis only, and does not correspond with that seen in OPNET's text editors.

```

. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .

1 /** state (FR_REPEAT) enter executives */
2 FSM_STATE_ENTER_FORCED (6, state6_enter_exec, "FR_REPEAT")
3 {
4 /* Extract the destination address of the frame. */
5 op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

6 /* If the frame is for this station, make a copy */
7 /* of the frame's data field and forward it to */
8 /* the higher layer. */
9 /* 14APR94 : In order to send the frames which are */
10 /* addressed to the remote lan, check the address database */
11 /* of remote lan. Frames addressed to the remote lan shouldn't */
12 /* be repeated in the local ring -- This is a simple forwarding */
13 /* decision algorithm, one of the bridge's function */
14 /* - Karayakaylar */
15 if((dest_addr == my_address)|| (dest_addr <= 9))
16 {
17 /* record total size of the frame (including data) */
18 pk_len = op_pk_total_size_get (pkptr);

19 /* decapsulate the data contents of the frame */
20 /* 29JAN94: a new field, "pri", has been added to */
21 /* the fddi_llc_fr packet format in the Parameters */
22 /* Editor, so that output statistics can be */
23 /* generated by class and priority. -Nix */
24 op_pk_nfd_get (pkptr, "info", &data_pkptr);
25 op_pk_nfd_get (pkptr, "pri", &pri_level);

26 /* The source and destination address are placed in the */
27 /* LLC's ICI before delivering the frame's contents. */
28 op_ici_attr_set (to_llc_ici_ptr, "src_addr", src_addr);

```

```

29 op_ici_attr_set (to_llc_ici_ptr, "dest_addr", dest_addr);
30 op_ici_install (to_llc_ici_ptr);

31 /* Because, as noted in the FR_RCV state, only the */
32 /* frame's leading edge has arrived at this time, the */
33 /* complete frame can only be delivered to the higher */
34 /* layer after the frame's transmission delay has elapsed. */
35 /* (since decapsulation of the frame data contents has occurred, */
36 /* the original MAC frame length is used to calculate delay) */
37 tx_time = (double) pk_len / FDDI_TX_RATE;
38 op_pk_send_delayed (data_pkptr, FDDI_LLC_STRM_OUT, tx_time);

39 /* Note that the standard specifies that the original */
40 /* frame should be passed along until the originating station */
41 /* receives it, at which point it is stripped from the ring. */
42 /* However, in the simulation model, there is no interest */
43 /* in letting the frame continue past its destination unless */
44 /* group addresses are used, so that the same frame could be */
45 /* destined for several stations. Here the frame is stripped */
46 /* for efficiency as it reaches the destination; if the model */
47 /* is modified to include group addresses, this should be changed */
48 /* so that the frame is copied and the original repeated. */
49 /* Logic is already present for stripping the frame at the origin. */
50 op_pk_destroy (pkptr);
51 }

52 /* 14APR94 : the frames belong to this ring should be repeated. */
53 /* Thus, local traffic is constrained.-- This is filtering decision */
54 /* One of the bridge's function - Karayakaylar */
55 else{
56 /* Repeat the original frame on the ring and account for */
57 /* the latency through the station and the propagation delay */
58 /* for a single hop. */
59 /* (Only the originating station can strip the frame). */
60 op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT,
61 Fddi_St_Latency + Fddi_Prop_Delay);
62 }
63 }

64 /** state (FR_REPEAT) exit executives **/
65 FSM_STATE_EXIT_FORCED (6, state6_exit_exec, "FR_REPEAT")
66 {
67 }
68 . . .
69 . . .
70 . . .
71 . . .
72 . . .
73 . . .
74 . . .

```

# APPENDIX F

## SPNI SINK "C" CODE

### "sp\_fddi\_sink.pr.c"

The line numbering in this appendix is within this thesis only, and does not correspond with that seen in OPNET's text editors.

```

1  /* Process model C form file: sp_fddi_sink.pr.c */
2  /* Portions of this file Copyright (C) MIL 3, Inc. 1992 */

3  /* OPNET system definitions */
4  #include <opnet.h>
5  #include "sp_fddi_sink.pr.h"
6  FSM_EXT_DECS

7  /* Header block */
8  /* Globals */
9  /* positions 0-7 represent the asynch priority levels, PRIORITIES + 1 */
10 /* represents synch traffic, and grand totals are as given in the original.
...
11 #define      PRIORITIES      8 /* 20JAN94 */
12 #define XMITTER_BUSY  0 /*10MAY94 */

13 static                                /* 05FEB94 */
14 double      fddi2_sink_accum_delay = 0.0;
15 static                                /* 05FEB94 */
16 double fddi2_sink_accum_delay_a[PRIORITIES + 1] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
17 static                                /* 05FEB94 */
18 int      fddi2_sink_total_pkts = 0;
19 static                                /* 05FEB94 */
20 int fddi2_sink_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
21 static                                /* 05FEB94 */
22 double      fddi2_sink_total_bits = 0.0;
23 static                                /* 05FEB94 */
24 double fddi2_sink_total_bits_a[PRIORITIES + 1] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
25 static                                /* 05FEB94 */
26 double      fddi2_sink_peak_delay = 0.0;
27 static                                /* 05FEB94 */
28 double fddi2_sink_peak_delay_a[PRIORITIES + 2] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
29 static                                /* 05FEB94 */
30 int fddi2_sink_scalar_write = 0;
31 static                                /* 05FEB94 */
32 int      pri2_set = 20;                /* 20JAN94 */

```

```

33 double          busy = 0.0;          /* 10MAY94 */

34 /* Statistics used for command link:21APR94 */
35 static
36 int              fddilp2_total_pkts = 0;
37 static
38 int fddilp2_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
39 static
40 double          fddilp2_total_bits = 0.0;
41 static
42 double fddilp2_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

43 /* Externally defined globals. */
44 extern double fddi_t_opr [];

45 /*12JAN94:attributes from the Environment file */
46 double  Offered_Load;          /* 12JAN94 */
47 double  Asynch_Offered_Load; /* 12JAN94 */

48 /* transition expressions */
49 #define END_OF_SIM  op_intrpt_type() == OPC_INTRPT_ENDSIM

50 /* State variable definitions */
51 typedef struct
52 {
53     FSM_SYS_STATE
54     Gshandle          sv_thru2_gshandle;
55     Gshandle          sv_m2_delay_gshandle;
56     Gshandle          sv_ete2_delay_gshandle;
57     Gshandle          sv_thru2_gshandle_a[10];
58     Gshandle          sv_m2_delay_gshandle_a[10];
59     Gshandle          sv_ete2_delay_gshandle_a[9];
60     Gshandle          sv_t2_gshandle;
61     Gshandle          sv_t2_gshandle_a[10];
62     Objid             sv_my_id;
63 } sp_fddi_sink_state;

64 #define pr_state_ptr          ((sp_fddi_sink_state*) SimI_Mod_State_Ptr)
65 #define thru2_gshandle        pr_state_ptr->sv_thru2_gshandle
66 #define m2_delay_gshandle     pr_state_ptr->sv_m2_delay_gshandle
67 #define ete2_delay_gshandle   pr_state_ptr->sv_ete2_delay_gshandle
68 #define thru2_gshandle_a      pr_state_ptr->sv_thru2_gshandle_a
69 #define m2_delay_gshandle_a    pr_state_ptr->sv_m2_delay_gshandle_a
70 #define ete2_delay_gshandle_a pr_state_ptr->sv_ete2_delay_gshandle_a
71 #define t2_gshandle           pr_state_ptr->sv_t2_gshandle
72 #define t2_gshandle_a         pr_state_ptr->sv_t2_gshandle_a
73 #define my_id                 pr_state_ptr->sv_my_id

```

```

74 /* Process model interrupt handling procedure */

75 void
76 sp_fddi_sink ()
77 {
78     double delay, creat_time;
79     Packet* pkptr;
80     Packet*      pkptr1 ; /*5APR94*/
81     int src_addr, my_addr;
82     int dest_addr; /*14APR94*/
83     Ici* from_mac_ici_ptr;
84     double fddi_sink_ttrt;

85     FSM_ENTER (sp_fddi_sink)

86     FSM_BLOCK_SWITCH
87     {
88         /*-----*/
89         /* state (DISCARD) enter executives */
90         FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "DISCARD")
91         {
92             /* determine the type of interrupt */
93             switch(op_intrpt_type())
94             {
95                 /* check if transmitter is busy */
96                 case OPC_INTRPT_STAT:
97                 {
98                     busy = op_stat_local_read (XMITTER_BUSY);
99                     break;
100                 }
101                 /* check if a packet has arrived */
102                 case OPC_INTRPT_STRM:
103                 {
104                     /* get the packet and the interface control info */
105                     pkptr = op_pk_get (op_intrpt_strm ());
106                     from_mac_ici_ptr = op_intrpt_ici ();

107                     /* 20JAN94: get the packet's priority level, which */
108                     /* will be used to index arrays of thruput and delay */
109                     /* computations. */
110                     /* pri2_set = op_pk_priority_get (pkptr); doesn't work here */
111                     op_pk_nfd_get (pkptr, "pri", &pri2_set); /* 29JAN94 */

112                     /* determine the time of creation of the packet */

```

```

113 op_pk_nfd_get (pkptr, "cr_time", &creat_time);

114 /* determine the dest address of the packet */ /*18APR94*/
115 op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

116 /* 7APR94:determine id of own processor to use in finding */
117 /* station address of the bridge node */
118 my_id = op_id_self();

119 /* 14APR94 : also get my own address */
120 op_ima_obj_attr_get ( my_id, "station_address", &my_addr);

121 /* destroy the packet */
122 /* op_pk_destroy (pkptr); */
123 /* 03FEB94: rather, enqueue the packet. This will be the */
124 /* first step toward developing a LAN bridging structure. */
125 /* -Nix */
126 /* op_subq_pk_insert (pri_set, pkptr, OPC_QPOS_TAIL); */

127 /* 14APR94: check the frame passed to "llc" is destined for */
128 /* this station. If it is destroy the packet; if not, allocate the packets */
129 /* to the command link transmitter since they are destined for the remote lan */
130 /* -Karayakaylar */
131 /* determine the packets coming from surface stations, this will */
132 /* be counted for local traffic */
133 /* 9(nine) is model specific, this is the "station_number" of */
134 /* collection platform bridge station */
135 if((dest_addr == my_addr)&&(src_addr > 9))
136 {
137 /* add in its size */
138 fddi2_sink_total_bits += op_pk_total_size_get (pkptr);
139 fddi2_sink_total_bits_a[pri2_set] += op_pk_total_size_get (pkptr); /* 20JAN-20APR94 */

140 /* accumulate delays */
141 delay = op_sim_time () - creat_time;
142 fddi2_sink_accum_delay += delay;
143 fddi2_sink_accum_delay_a[pri2_set] += delay; /* 20JAN-20APR94 */

144 /* keep track of peak delay value */
145 if (delay > fddi2_sink_peak_delay)
146 fddi2_sink_peak_delay = delay;

147 /* 20JAN94: keep track by priority levels as well 23JAN-20APR94 */
148 if (delay > fddi2_sink_peak_delay_a[pri2_set])
149     fddi2_sink_peak_delay_a[pri2_set] = delay;

150 op_pk_destroy (pkptr);

151 /* increment packet counter; 20JAN94 */
152 fddi2_sink_total_pkts++;
153 fddi2_sink_total_pkts_a[pri2_set]++;

```

```

154 /* if a multiple of 25 packets is reached, update stats */
155 /* 03FEB94: [0]->[7] represent asynch priorities 1->8, */
156 /* respectively; [8] represents synchronous traffic, */
157 /* and [9] represents overall asynchronous traffic.-Nix */
158 if (fddi2_sink_total_pkts % 25 == 0)
159 {
160   op_stat_global_write (thru2_gshandle,
161     fddi2_sink_total_bits / op_sim_time ());

162   op_stat_global_write (thru2_gshandle_a[pri2_set],
163     fddi2_sink_total_bits_a[0] / op_sim_time());
164   op_stat_global_write (thru2_gshandle_a[0],
165     fddi2_sink_total_bits_a[1] / op_sim_time());
166   op_stat_global_write (thru2_gshandle_a[1],
167     fddi2_sink_total_bits_a[pri2_set] / op_sim_time());
168   op_stat_global_write (thru2_gshandle_a[2],
169     fddi2_sink_total_bits_a[2] / op_sim_time());
170   op_stat_global_write (thru2_gshandle_a[3],
171     fddi2_sink_total_bits_a[3] / op_sim_time());
172   op_stat_global_write (thru2_gshandle_a[4],
173     fddi2_sink_total_bits_a[4] / op_sim_time());
174   op_stat_global_write (thru2_gshandle_a[5],
175     fddi2_sink_total_bits_a[5] / op_sim_time());
176   op_stat_global_write (thru2_gshandle_a[6],
177     fddi2_sink_total_bits_a[6] / op_sim_time());
178   op_stat_global_write (thru2_gshandle_a[7],
179     fddi2_sink_total_bits_a[7] / op_sim_time());
180   op_stat_global_write (thru2_gshandle_a[8],
181     fddi2_sink_total_bits_a[8] / op_sim_time());
182
183   /* 30JAN94: gather all asynch stats into one overall figure */
184   op_stat_global_write (thru2_gshandle_a[9],
185     (fddi2_sink_total_bits - fddi2_sink_total_bits_a[8]) /
186     op_sim_time());

187   /* (fddi2_sink_total_bits_a[0] + fddi2_sink_total_bits_a[1] + */
188   /* fddi2_sink_total_bits_a[2] + fddi2_sink_total_bits_a[3] + */
189   /* fddi2_sink_total_bits_a[4] + fddi2_sink_total_bits_a[5] + */
190   /* fddi2_sink_total_bits_a[6] + fddi2_sink_total_bits_a[7]) / */
191   /* op_sim_time()); */

192   op_stat_global_write (m2_delay_gshandle,
193     fddi2_sink_accum_delay / fddi2_sink_total_pkts);

194   op_stat_global_write (m2_delay_gshandle_a[0],
195     fddi2_sink_accum_delay_a[0] / fddi2_sink_total_pkts_a[0]);
196   op_stat_global_write (m2_delay_gshandle_a[1],
197     fddi2_sink_accum_delay_a[1] / fddi2_sink_total_pkts_a[1]);

```

```

197 op_stat_global_write (m2_delay_gshandle_a[2],
198 fddi2_sink_accum_delay_a[2] / fddi2_sink_total_pkts_a[2]);
199 op_stat_global_write (m2_delay_gshandle_a[3],
200 fddi2_sink_accum_delay_a[3] / fddi2_sink_total_pkts_a[3]);
201 op_stat_global_write (m2_delay_gshandle_a[4],
202 fddi2_sink_accum_delay_a[4] / fddi2_sink_total_pkts_a[4]);
203 op_stat_global_write (m2_delay_gshandle_a[5],
204 fddi2_sink_accum_delay_a[5] / fddi2_sink_total_pkts_a[5]);
205 op_stat_global_write (m2_delay_gshandle_a[6],
206 fddi2_sink_accum_delay_a[6] / fddi2_sink_total_pkts_a[6]);
207 op_stat_global_write (m2_delay_gshandle_a[7],
208 fddi2_sink_accum_delay_a[7] / fddi2_sink_total_pkts_a[7]);
209 op_stat_global_write (m2_delay_gshandle_a[8],
210 fddi2_sink_accum_delay_a[8] / fddi2_sink_total_pkts_a[8]);

211 /* 30JAN94: gather all asynch stats into one figure */
212 op_stat_global_write (m2_delay_gshandle_a[9],
213 (fddi2_sink_accum_delay - fddi2_sink_accum_delay_a[8]) /
214 (fddi2_sink_total_pkts - fddi2_sink_total_pkts_a[8]));

215 /* (fddi2_sink_accum_delay_a[0] + fddi2_sink_accum_delay_a[1] + */
216 /* fddi2_sink_accum_delay_a[2] + fddi2_sink_accum_delay_a[3] + */
217 /* fddi2_sink_accum_delay_a[4] + fddi2_sink_accum_delay_a[5] + */
218 /* fddi2_sink_accum_delay_a[6] + fddi2_sink_accum_delay_a[7]) / */
219 /* (fddi2_sink_total_pkts_a[0] + fddi2_sink_total_pkts_a[1] + */
220 /* fddi2_sink_total_pkts_a[2] + fddi2_sink_total_pkts_a[3] + */
221 /* fddi2_sink_total_pkts_a[4] + fddi2_sink_total_pkts_a[5] + */
222 /* fddi2_sink_total_pkts_a[6] + fddi2_sink_total_pkts_a[7])); */

223 /* also record actual delay values */
224 op_stat_global_write (ete2_delay_gshandle, delay);
225 op_stat_global_write (ete2_delay_gshandle_a[pri2_set], delay);
226 }
227 }/*end of if(dest_addr==my_addr)&&(src_addr > 9)statement */

228 /* 20APR94: destroy the packets coming from the first lan destined */
229 /* for this station. These packets are not counted for local traffic.*/
230 else if(dest_addr == my_addr)
231 op_pk_destroy(pkptr);

232 /* Other frames passed to "llc" should be destined for other lan */
233 /* 18APR94 :allocate the packets to transmitter of command link */
234 else
235 {

236 /* add in its size */
237 fddilp2_total_bits += op_pk_total_size_get (pkptr);
238 fddilp2_total_bits_a[pri2_set] += op_pk_total_size_get (pkptr); /* 20JAN-20APR94 */

```



```

239 /* increment packet counter; 20APR94 */
240 fddilp2_total_pkts++;
241 fddilp2_total_pkts_a[pri2_set]++;

242 /* if a multiple of 25 packets is reached, update stats */
243 /* [0]->[7] represent asynch priorities 1->8, */
244 /* respectively; [8] represents synchronous traffic, */
245 /* and [9] represents overall asynchronous traffic.-Wix */
246 if (fddilp2_total_pkts % 25 == 0)
247 {
248   op_stat_global_write (t2_gshandle,
249     fddilp2_total_bits / op_sim_time ());

250   op_stat_global_write (t2_gshandle_a[pri2_set],
251     fddilp2_total_bits_a[0] / op_sim_time());
252   op_stat_global_write (t2_gshandle_a[0],
253     fddilp2_total_bits_a[1] / op_sim_time());
254   op_stat_global_write (t2_gshandle_a[1],
255     fddilp2_total_bits_a[pri2_set] / op_sim_time());
256   op_stat_global_write (t2_gshandle_a[2],
257     fddilp2_total_bits_a[2] / op_sim_time());
258   op_stat_global_write (t2_gshandle_a[3],
259     fddilp2_total_bits_a[3] / op_sim_time());
260   op_stat_global_write (t2_gshandle_a[4],
261     fddilp2_total_bits_a[4] / op_sim_time());
262   op_stat_global_write (t2_gshandle_a[5],
263     fddilp2_total_bits_a[5] / op_sim_time());
264   op_stat_global_write (t2_gshandle_a[6],
265     fddilp2_total_bits_a[6] / op_sim_time());
266   op_stat_global_write (t2_gshandle_a[7],
267     fddilp2_total_bits_a[7] / op_sim_time());
268   op_stat_global_write (t2_gshandle_a[8],
269     fddilp2_total_bits_a[8] / op_sim_time());

270 /* gather all asynch stats into one overall figure */
271   op_stat_global_write (t2_gshandle_a[9],
272     (fddilp2_total_bits - fddilp2_total_bits_a[8]) /
273     op_sim_time());

274                                     /* (fddilp2_total_bits_a[0] + fddilp2_total_bits_a[1] +
275   /* fddilp2_total_bits_a[2] + fddilp2_total_bits_a[3] + */
276   /* fddilp2_total_bits_a[4] + fddilp2_total_bits_a[5] + */
277   /* fddilp2_total_bits_a[6] + fddilp2_total_bits_a[7]) / */
278   /* op_sim_time()); */

279 }

280 /* 21APR94:allocate packets to the command link transmitter */

```

```

281 op_subq_pk_insert(0, pkptr, OPC_QPOS_TAIL);

282 /* check if this subqueue is empty and transmitter is not busy */
283 if ((!op_subq_empty(0)) && (busy == 0.0))
284 {
285 /*access the first packet in the subqueue */
286 pkptr1 = op_subq_pk_remove (0, OPC_QPOS_HEAD);
287 /* forward it to the transmitter of command link */
288 op_pk_send (pkptr1, 0);
289 }
290 /* end of else */
291 break;
292 /* end of case OPC_INTRPT_STRM statement */

293 /* end of switch */

294 }

295 /** blocking after enter executives of unforced state. **/
296 FSM_EXIT (1, sp_fddi_sink)

297 /** state (DISCARD) exit executives **/
298 FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "DISCARD")
299 {
300 }

301 /** state (DISCARD) transition processing **/
302 FSM_INIT_COND (END_OF_SIM)
303 FSM_DFLT_COND
304 FSM_TEST_LOGIC ("DISCARD")

305 FSM_TRANSIT_SWITCH
306 {
307 FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
308 FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
309 }
310 /*-----*/

311 /** state (STATS) enter executives **/
312 FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "STATS")
313 {
314 /* At end of simulation, scalar performance statistics */
315 /* and input parameters are written out. */
316 /* This is for command link throughput :21APR94*/
317 op_stat_scalar_write ("CL Throughput (bps), Priority 1",
318 fddilp2_total_bits_a[0] / op_sim_time ());

```

```

319 op_stat_scalar_write ("CL Throughput (bps), Priority 2",
320     fddilp2_total_bits_a[1] / op_sim_time ());

321 op_stat_scalar_write ("CL Throughput (bps), Priority 3",
322     fddilp2_total_bits_a[2] / op_sim_time ());

323 op_stat_scalar_write ("CL Throughput (bps), Priority 4",
324     fddilp2_total_bits_a[3] / op_sim_time ());

325 op_stat_scalar_write ("CL Throughput (bps), Priority 5",
326     fddilp2_total_bits_a[4] / op_sim_time ());

327 op_stat_scalar_write ("CL Throughput (bps), Priority 6",
328     fddilp2_total_bits_a[5] / op_sim_time ());

329 op_stat_scalar_write ("CL Throughput (bps), Priority 7",
330     fddilp2_total_bits_a[6] / op_sim_time ());

331 op_stat_scalar_write ("CL Throughput (bps), Priority 8",
332     fddilp2_total_bits_a[7] / op_sim_time ());

333 op_stat_scalar_write ("CL Throughput (bps), Asynchronous",
334     (fddilp2_total_bits - fddilp2_total_bits_a[8]) / op_sim_time ());

335 /* (fddilp2_total_bits_a[0] + fddilp2_total_bits_a[1] + */
336 /* fddilp2_total_bits_a[2] + fddilp2_total_bits_a[3] + */
337 /* fddilp2_total_bits_a[4] + fddilp2_total_bits_a[5] + */
338 /* fddilp2_total_bits_a[6] + fddilp2_total_bits_a[7]) / */
339 /* op_sim_time ()); */

340 op_stat_scalar_write ("CL Throughput (bps), Synchronous",
341     fddilp2_total_bits_a[8] / op_sim_time ());

342 op_stat_scalar_write ("CL Throughput (bps), Total",
343     fddilp2_total_bits / op_sim_time ());

344 /* Only one station needs to do this for the second ring(Ring 1)*/
345 if (!fddi2_sink_scalar_write)
346 {
347 /* set the scalar write flag */
348 fddi2_sink_scalar_write = 1;

349 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 1",
350 fddi2_sink_accum_delay_a[0] / fddi2_sink_total_pkts_a[0]);

```

```

351 op_stat_scalar_write ("Mean End-to-End Delay-1(sec.), Priority 2",
352     fddi2_sink_accum_delay_a[1] / fddi2_sink_total_pkts_a[1]);

353 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 3",
354     fddi2_sink_accum_delay_a[2] / fddi2_sink_total_pkts_a[2]);

355 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 4",
356     fddi2_sink_accum_delay_a[3] / fddi2_sink_total_pkts_a[3]);

357 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 5",
358     fddi2_sink_accum_delay_a[4] / fddi2_sink_total_pkts_a[4]);

359 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 6",
360     fddi2_sink_accum_delay_a[5] / fddi2_sink_total_pkts_a[5]);

361 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 7",
362     fddi2_sink_accum_delay_a[6] / fddi2_sink_total_pkts_a[6]);

363 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 8",
364     fddi2_sink_accum_delay_a[7] / fddi2_sink_total_pkts_a[7]);

365 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Asynchronous",
366     (fddi2_sink_accum_delay - fddi2_sink_accum_delay_a[8]) /
367     (fddi2_sink_total_pkts - fddi2_sink_total_pkts_a[8]));

368 /*      (fddi2_sink_accum_delay_a[0] + fddi2_sink_accum_delay_a[1] + */
369 /*      fddi2_sink_accum_delay_a[2] + fddi2_sink_accum_delay_a[3] + */
370 /*      fddi2_sink_accum_delay_a[4] + fddi2_sink_accum_delay_a[5] + */
371 /*      fddi2_sink_accum_delay_a[6] + fddi2_sink_accum_delay_a[7]) / */
372 /*      (fddi2_sink_total_pkts_a[0] + fddi2_sink_total_pkts_a[1] + */
373 /*      fddi2_sink_total_pkts_a[2] + fddi2_sink_total_pkts_a[3] + */
374 /*      fddi2_sink_total_pkts_a[4] + fddi2_sink_total_pkts_a[5] + */
375 /*      fddi2_sink_total_pkts_a[6] + fddi2_sink_total_pkts_a[7])); */

376 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Synchronous",
377     fddi2_sink_accum_delay_a[8] / fddi2_sink_total_pkts_a[8]);

378 op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Total",
379     fddi2_sink_accum_delay / fddi2_sink_total_pkts);

380 op_stat_scalar_write ("Throughput-1 (bps), Priority 1",
381     fddi2_sink_total_bits_a[0] / op_sim_time ());

382 op_stat_scalar_write ("Throughput-1 (bps), Priority 2",
383     fddi2_sink_total_bits_a[1] / op_sim_time ());

384 op_stat_scalar_write ("Throughput-1 (bps), Priority 3",
385     fddi2_sink_total_bits_a[2] / op_sim_time ());

386 op_stat_scalar_write ("Throughput-1( bps), Priority 4",

```

```

387         fddi2_sink_total_bits_a[3] / op_sim_time ());

388     op_stat_scalar_write ("Throughput-1 (bps), Priority 5",
389         fddi2_sink_total_bits_a[4] / op_sim_time ());

390     op_stat_scalar_write ("Throughput-1 (bps), Priority 6",
391         fddi2_sink_total_bits_a[5] / op_sim_time ());

392     op_stat_scalar_write ("Throughput-1 (bps), Priority 7",
393         fddi2_sink_total_bits_a[6] / op_sim_time ());

394     op_stat_scalar_write ("Throughput-1 (bps), Priority 8",
395         fddi2_sink_total_bits_a[7] / op_sim_time ());

396     op_stat_scalar_write ("Throughput-1 (bps), Asynchronous",
397         (fddi2_sink_total_bits - fddi2_sink_total_bits_a[8]) / op_sim_time ());

398     /*      (fddi2_sink_total_bits_a[0] + fddi2_sink_total_bits_a[1] + */
399     /*      fddi2_sink_total_bits_a[2] + fddi2_sink_total_bits_a[3] + */
400     /*      fddi2_sink_total_bits_a[4] + fddi2_sink_total_bits_a[5] + */
401     /*      fddi2_sink_total_bits_a[6] + fddi2_sink_total_bits_a[7]) / */
402     /*      op_sim_time ()); */

403     op_stat_scalar_write ("Throughput-1 (bps), Synchronous",
404         fddi2_sink_total_bits_a[8] / op_sim_time ());

405     op_stat_scalar_write ("Throughput-1 (bps), Total",
406         fddi2_sink_total_bits / op_sim_time ());

407 op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 1",
408 fddi2_sink_peak_delay_a[0]);

409     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 2",
410         fddi2_sink_peak_delay_a[1]);

411     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 3",
412         fddi2_sink_peak_delay_a[2]);

413     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 4",
414         fddi2_sink_peak_delay_a[3]);

415     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 5",
416         fddi2_sink_peak_delay_a[4]);

417     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 6",
418         fddi2_sink_peak_delay_a[5]);

```

```

419     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 7",
420         fddi2_sink_peak_delay_a[6]);

421     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 8",
422         fddi2_sink_peak_delay_a[7]);

423     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Synchronous",
424         fddi2_sink_peak_delay_a[8]);

425     op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Overall",
426         fddi2_sink_peak_delay);

427 /* Write the TTRT value for ring 0. This preserves */
428 /* the old behavior for single-ring simulations. */
429 op_stat_scalar_write ("TTRT (sec.) - Ring 1",
430     fddi_t_opr [1]);

431 /* 12JAN94: obtain offered load information from the Environment */
432 /* file; this will be used to provide abscissa information that */
433 /* can be plotted in the Analysis Editor (see "fddi_sink" STATS */
434 /* state. To the user: it's your job to keep these current in */
435 /* the Environment File. -Mlx */
436 op_ima_sim_attr_get (OPC_IMA_DOUBLE, "total_offered_load_1", &Offered_Load);
437 op_ima_sim_attr_get (OPC_IMA_DOUBLE, "asynch_offered_load_1", &Asynch_Offered_Load);

438     /* 12JAN94: write the total offered load for this run */
439     op_stat_scalar_write ("Total Offered Load-1 (Mbps)",
440         Offered_Load);

441     op_stat_scalar_write ("Asynchronous Offered Load-1 (Mbps)",
442         Asynch_Offered_Load);
443 }
444 }

445 /** blocking after enter executives of unforced state. **/
446 FSM_EXIT (3, sp_fddi_sink)

447 /** state (STATS) exit executives **/
448 FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "STATS")
449 {
450 }

451 /** state (STATS) transition processing **/
452 FSM_TRANSIT_MISSING ("STATS")
453 /*-----*/

```

```

454 /** state (INIT) enter executives **/
455 FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "INIT")
456 {
457 /* get the gshandles of the global statistic to be obtained */
458 /* 20JAN94: set array format */

459 thru2_gshandle_a[0] = op_stat_global_reg ("pri 1 throughput-1 (bps)");
460 thru2_gshandle_a[1] = op_stat_global_reg ("pri 2 throughput-1 (bps)");
461 thru2_gshandle_a[2] = op_stat_global_reg ("pri 3 throughput-1 (bps)");
462 thru2_gshandle_a[3] = op_stat_global_reg ("pri 4 throughput-1 (bps)");
463 thru2_gshandle_a[4] = op_stat_global_reg ("pri 5 throughput-1 (bps)");
464 thru2_gshandle_a[5] = op_stat_global_reg ("pri 6 throughput-1 (bps)");
465 thru2_gshandle_a[6] = op_stat_global_reg ("pri 7 throughput-1 (bps)");
466 thru2_gshandle_a[7] = op_stat_global_reg ("pri 8 throughput-1 (bps)");
467 thru2_gshandle_a[8] = op_stat_global_reg ("synch throughput-1 (bps)");
468 thru2_gshandle_a[9] = op_stat_global_reg ("async throughput-1 (bps)");
469 thru2_gshandle      = op_stat_global_reg ("total throughput-1 (bps)");

470 m2_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 mean delay-1 (sec.)");
471 m2_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 mean delay-1 (sec.)");
472 m2_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 mean delay-1 (sec.)");
473 m2_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 mean delay-1 (sec.)");
474 m2_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 mean delay-1 (sec.)");
475 m2_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 mean delay-1 (sec.)");
476 m2_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 mean delay-1 (sec.)");
477 m2_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 mean delay-1 (sec.)");
478 m2_delay_gshandle_a[8] = op_stat_global_reg ("synch mean delay-1 (sec.)");
479 m2_delay_gshandle_a[9] = op_stat_global_reg ("async mean delay-1 (sec.)");
480 m2_delay_gshandle      = op_stat_global_reg ("total mean delay-1 (sec.)");

481 ete2_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 end-to-end delay-1 (sec.)");
482 ete2_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 end-to-end delay-1 (sec.)");
483 ete2_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 end-to-end delay-1 (sec.)");
484 ete2_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 end-to-end delay-1 (sec.)");
485 ete2_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 end-to-end delay-1 (sec.)");
486 ete2_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 end-to-end delay-1 (sec.)");
487 ete2_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 end-to-end delay-1 (sec.)");
488 ete2_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 end-to-end delay-1 (sec.)");
489 ete2_delay_gshandle_a[8] = op_stat_global_reg ("synch end-to-end delay-1 (sec.)");
490 ete2_delay_gshandle      = op_stat_global_reg ("total end-to-end delay-1 (sec.)");

491 t2_gshandle_a[0] = op_stat_global_reg ("pri 1 CL throughput (bps)");
492 t2_gshandle_a[1] = op_stat_global_reg ("pri 2 CL throughput (bps)");
493 t2_gshandle_a[2] = op_stat_global_reg ("pri 3 CL throughput (bps)");
494 t2_gshandle_a[3] = op_stat_global_reg ("pri 4 CL throughput (bps)");
495 t2_gshandle_a[4] = op_stat_global_reg ("pri 5 CL throughput (bps)");
496 t2_gshandle_a[5] = op_stat_global_reg ("pri 6 CL throughput (bps)");
497 t2_gshandle_a[6] = op_stat_global_reg ("pri 7 CL throughput (bps)");
498 t2_gshandle_a[7] = op_stat_global_reg ("pri 8 CL throughput (bps)");

```

```

499 t2_gshandle_a[8] = op_stat_global_reg ("synch CL throughput (bps)");
500 t2_gshandle_a[9] = op_stat_global_reg ("async CL throughput (bps)");
501 t2_gshandle      = op_stat_global_reg ("total CL throughput (bps)");

502 }

503 /** state (INIT) exit executives **/
504 FSM_STATE_EXIT_FORCED (2, state2_exit_exec, "INIT")
505 {
506 }

507 /** state (INIT) transition processing **/
508 FSM_INIT_COND (END_OF_SIM)
509 FSM_DFLT_COND
510 FSM_TEST_LOGIC ("INIT")

511 FSM_TRANSIT_SWITCH
512 {
513   FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
514   FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
515 }
516 /*-----*/

517 }

518 FSM_EXIT (2, sp_fddi_sink)
519 }

520 void
521 sp_fddi_sink_svar (prs_ptr, var_name, var_p_ptr)
522 sp_fddi_sink_state *prs_ptr;
523 char *var_name, **var_p_ptr;
524 {

525   FIN (sp_fddi_sink_svar (prs_ptr))

526   *var_p_ptr = VOS_NIL;
527   if (Vos_String_Equal ("thru2_gshandle" , var_name))
528     *var_p_ptr = (char *) (&prs_ptr->sv_thru2_gshandle);
529   if (Vos_String_Equal ("m2_delay_gshandle" , var_name))
530     *var_p_ptr = (char *) (&prs_ptr->sv_m2_delay_gshandle);
531   if (Vos_String_Equal ("ete2_delay_gshandle" , var_name))

```



```

532 *var_p_ptr = (char *) (&prs_ptr->sv_ete2_delay_gshandle);
533 if (Vos_String_Equal ("thru2_gshandle_a" , var_name))
534 *var_p_ptr = (char *) (prs_ptr->sv_thru2_gshandle_a);
535 if (Vos_String_Equal ("m2_delay_gshandle_a" , var_name))
536 *var_p_ptr = (char *) (prs_ptr->sv_m2_delay_gshandle_a);
537 if (Vos_String_Equal ("ete2_delay_gshandle_a" , var_name))
538 *var_p_ptr = (char *) (prs_ptr->sv_ete2_delay_gshandle_a);
539 if (Vos_String_Equal ("t2_gshandle" , var_name))
540 *var_p_ptr = (char *) (&prs_ptr->sv_t2_gshandle);
541 if (Vos_String_Equal ("t2_gshandle_a" , var_name))
542 *var_p_ptr = (char *) (prs_ptr->sv_t2_gshandle_a);
543 if (Vos_String_Equal ("my_id" , var_name))
544 *var_p_ptr = (char *) (&prs_ptr->sv_my_id);

```

```

545 FOOT;
546 }

```

```

547 void
548 sp_fddi_sink_diag ()
549 {
550 double delay, creat_time;
551 Packet* pkptr;
552 Packet*      pkptr1 ; /*5APR94*/
553 int src_addr, my_addr;
554 int dest_addr; /*14APR94*/
555 Ici* from_mac_ici_ptr;
556 double fddi_sink_ttrt;

```

```

557 FIN (sp_fddi_sink_diag ())

```

```

558 FOOT;
559 }

```

```

560 void
561 sp_fddi_sink_terminate ()
562 {
563 double delay, creat_time;
564 Packet* pkptr;
565 Packet*      pkptr1 ; /*5APR94*/
566 int src_addr, my_addr;
567 int dest_addr; /*14APR94*/
568 Ici* from_mac_ici_ptr;
569 double fddi_sink_ttrt;

```

```

570 FIN (sp_fddi_sink_terminate ())

571 FOUT;
572 }

573 Compcode
574 sp_fddi_sink_init (pr_state_pptr)
575 sp_fddi_sink_state **pr_state_pptr;
576 {
577 static VosT_Cm_Obtype obtype = OPC_NIL;

578 FIN (sp_fddi_sink_init (pr_state_pptr))

579 if (obtype == OPC_NIL)
580 {
581 if (Vos_Catmem_Register ("proc state vars (sp_fddi_sink)",
582 sizeof (sp_fddi_sink_state), Vos_Nop, &obtype) == VOSC_FAILURE)
583 FRET (OPC_COMPCODE_FAILURE)
584 }

585 if ((*pr_state_pptr = (sp_fddi_sink_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
586 FRET (OPC_COMPCODE_FAILURE)
587 else
588 {
589 (*pr_state_pptr)->current_block = 4;
590 FRET (OPC_COMPCODE_SUCCESS)
591 }
592 }

```

## APPENDIX G

### CDL MODEL ERROR ALLOCATION CODE

"cdl\_pt\_error.ps.c"

The line numbering in this appendix is within this thesis only, and does not correspond with that seen in OPNET's text editors.

```

1  /* cdl_pt_error.ps.c */
2  /* Customized error allocation model for point-to-point link transceiver pipeline */
3  /*****
4  /*      Last modified by Selcuk Karayakaylar      */
5  /*      22APR94      */
6  *****/

7  #include <opnet.h>
8  #include <math.h>

9  /* Define a convenient macro for computing */
10 /* factorials using the gamma function */
11 #define log_factorial(n) lgamma ((double) n + 1.0)

12 void
13 cdl_pt_error (pkptr)
14     Packet*   pkptr;
15     {
16         Objid      link_objid;
17         double      pe, r, p_accum, p_exact;
18         double      log_p1, log_p2, log_arrange;
19         double      duty_cycle; /* 31MAR94 */
20         double      jam_length, jam_ber, int_bet_jamlen, ber_bet_jamlen; /*29MAR94*/
21         double      time_stamp; /* time stamp for the packet arriving time */
22         double      offset; /* 1APR94 */
23         int         invert_errors = OPC_FALSE, seg_size, num_errs;
24         int         jammer_type; /*26APR94*/
25         /*int        channel_index; /* 4APR94 */

26         /* Compute the number of errors assigned to the given packet */
27         /* based on its length and the bit error probability. */
28         FIN (cdl_pt_error (pkptr))

29         /*Make a time stamp to see whether the packet is in jamming period or not */
30         time_stamp = op_sim_time();
31         /*printf("time_stamp = %16.12f\n", time_stamp);*/

32         /* Obtain object id of point-to-point link carrying transmission. */
33         link_objid = op_td_get_int (pkptr, OPC_TDA_PT_LINK_OBJID);

```

```

34  /* Obtain the channel index for the particular link */
35  /* Determine which channel the packet is on *//*30MAR*/
36  /* channel_index = op_td_get_int (pkptr,OPC_TDA_PT_CH_INDEX);*//*4APR94*/

37  /* Obtain the bit-error probability of the channel. */
38  /* op_ima_obj_attr_get (link_objid, "ber", &pe); *//*ignore this attribute 31MAR94*/

39  /* Obtain the extended attributes for the point-to-point link */
40  /* These attributes are appended in order to simulate jamming features */
41  /* 29MAR94 */

42  op_ima_obj_attr_get (link_objid, "jam_length", &jam_length);
43  op_ima_obj_attr_get (link_objid, "jam_ber", &jam_ber);
44  op_ima_obj_attr_get (link_objid, "interval_bet_jam_len", &int_bet_jamlen);
45  op_ima_obj_attr_get (link_objid, "ber_bet_jam_len", &ber_bet_jamlen);
46  op_ima_obj_attr_get (link_objid, "init_jam_offset", &offset);
47  op_ima_obj_attr_get (link_objid, "jammer_type", &jammer_type);

48  /* Obtain the length of the packet. */
49  seg_size = op_pk_total_size_get (pkptr);

50  /* Determine the jammer type in use:26APR94 */
51  /* Check if pulsed jammer is in use */
52  if (jammer_type == 0)
53      {
54          /* Randomize the jamming durations */
55          /* These durations are randomized with uniform distribution */
56          /* in range [0, duration[. User should be aware of these */
57          /* attributes specified in the environment file. They are max values */
58          /* for those particular durations */
59          jam_length = op_dist_uniform(jam_length);
60          int_bet_jamlen = op_dist_uniform(int_bet_jamlen);
61      }
62  /* Otherwise, channel swept jammer is in use. Jamming durations */
63  /* should not be randomized to keep consecutive pulses in order. */

64  /* Compute duty cycle for jamming */
65  duty_cycle = jam_length + int_bet_jamlen;

66  /* Check time stamp if it is in the initial jam offset period */
67  /* All BER's are uniformly distributed in range [0, ber[, so that */
68  /* realistic representation is provided; User should be aware of */
69  /* these attributes specified in the environment file. They are max values */
70  /* for those particular bers*/
71  if (time_stamp < offset)
72      pe = op_dist_uniform(ber_bet_jamlen); /* the packet is still not in the jamming */
73                                         /* period */
74  else
75      {
76          /* Check packet is in jamming period */

```

```

77     if ( fmod(time_stamp,duty_cycle) <= jam_length )
78         pe = op_dist_uniform(jam_ber); /* the packet is in jamming period, */
79         /* random "pe" to be computed as jam_ber */
80     else
81         pe = op_dist_uniform(ber_bet_jamlen); /* packet is in unjammed period */
82         /* random "pe" to be computed as ber_bet_jamlen */
83     }

84     /*This part computes num_errs for the packet */
85     /* Case 1: if the bit error rate is zero, so is the number of errors. */
86     if (pe == 0.0 || seg_size == 0)
87         num_errs = 0;

88     /* Case 2: if the bit error rate is 1.0, then all the bits are in error.*/
89     /* (note however, that bit error rates should not normally exceed 0.5).*/
90     else if (pe >= 1.0)
91         num_errs = seg_size;

92     /* Case 3: The bit error rate is not zero or one. */
93     else
94     {
95         /* If the bit error rate is greater than 0.5 and less than 1.0, invert */
96         /* the problem to find instead the number of bits that are not in error */
97         /* in order to accelerate the performance of the algorithm. Set a flag */
98         /* to indicate that the result will then have to be inverted. */
99         if (pe > 0.5)
100         {
101             pe = 1.0 - pe;
102             invert_errors = OPC_TRUE;
103         }

104         /* The error count can be obtained by mapping a uniform random number */
105         /* in [0, 1[ via the inverse of the cumulative mass function (CMF) */
106         /* for the bit error count distribution. */

107         /* Obtain a uniform random number in [0, 1[ to represent */
108         /* the value of the CDF at the outcome that will be produced. */
109         r = op_dist_uniform (1.0);

110         /* Integrate probability mass over possible outcomes until r is exceeded. */
111         /* The loop iteratively corresponds to "inverting" the CMF since it finds */
112         /* the bit error count at which the CMF first meets or exceeds the value r. */
113         for (p_accum = 0.0, num_errs = 0; num_errs <= seg_size; num_errs++)
114         {
115             /* Compute the probability of exactly 'num_errs' bit errors occurring. */

116             /* The probability that the first 'num_errs' bits will be in error */
117             /* is given by pow (pe, num_errs). Here it is obtained in logarithmic */
118             /* form to avoid underflow for small 'pe' or large 'num_errs_jam'. */
119             log_p1 = (double) num_errs * log (pe);

```

```

120      /* Similarly, obtain the probability that the remaining bits will not */
121      /* be in error. The combination of these two events represents one */
122      /* possible configuration of bits yielding a total of 'num_errs' errors. */
123      log_p2 = (double) (seg_size - num_errs) * log (1.0 - pe);

124      /* Compute the number of arrangements that are possible with the same */
125      /* number of bits in error as the particular case above. Again obtain */
126      /* this number in logarithmic form (to avoid overflow in this case). */
127      /* This result is expressed as the logarithmic form of the formula for */
128      /* the number N of combinations of k items from n:  $N = n!/(n-k)!k!$  */
129      log_arrange = log_factorial (seg_size) -
130                  log_factorial (num_errs) -
131                  log_factorial (seg_size - num_errs);
132      /* Compute the probability that exactly 'num_errs' are present */
133      /* in the segment of bits, in any arrangement. */
134      p_exact = exp (log_arrange + log_p1 + log_p2);

135      /* Add this to the probability mass accumulated so far for previously */
136      /* tested outcomes to obtain the value of the CMF at outcome=num_errs*/
137      p_accum += p_exact;

138      /* 'num_errs' is the outcome for this trial if the CMF meets or exceeds */
139      /* the uniform random value selected earlier. */
140      if (p_accum >= r)
141          break;
142      }

143      /* If the bit error rate was inverted to compute correct bits instead, then */
144      /* reinvert the result to obtain the number of bits in error. */
145      if (invert_errors == OPC_TRUE)
146          num_errs = seg_size - num_errs;
147      }

148      /* printf("num_of_errors = %5d\n", num_errs); */
149      /* Set number of bit errors in packet transmission data attribute. */
150      op_td_set_int (pkptr, OPC_TDA_PT_NUM_ERRORS, num_errs);
151      FOUT
152      }

```

# APPENDIX H

## SAMPLE ENVIRONMENT FILE FOR PULSED JAMMER

```
# cd14_lb0jam0.ef
# sample simulation configuration file for
# two interconnected 10 station network in the
# existence of pulsed jammer interference (137.088 Mbps channel hierarchy )
# with circular allocation load balancing algorithm

*** Attributes related to loading used by "fddi_gen" ***

# station addresses
*.ring0.f0.mac.station_address: 0
*.ring0.f1.mac.station_address: 1
*.ring0.f2.mac.station_address: 2
*.ring0.f3.mac.station_address: 3
*.ring0.f4.mac.station_address: 4
*.ring0.f5.mac.station_address: 5
*.ring0.f6.mac.station_address: 6
*.ring0.f7.mac.station_address: 7
*.ring0.f8.mac.station_address: 8
*.ring0.f9.mac.station_address: 9

*.ring1.f0.mac.station_address: 10
*.ring1.f1.mac.station_address: 11
*.ring1.f2.mac.station_address: 12
*.ring1.f3.mac.station_address: 13
*.ring1.f4.mac.station_address: 14
*.ring1.f5.mac.station_address: 15
*.ring1.f6.mac.station_address: 16
*.ring1.f7.mac.station_address: 17
*.ring1.f8.mac.station_address: 18
*.ring1.f9.mac.station_address: 19

*.ring0.*.mac.ring_id :0
*.ring1.*.mac.ring_id :1

# Specific stations may be tailored by specifying the full name:
# for example, top.ring0.f19.llc_src.async_mix : .5
# This means all stations must be specified, or individuals
# may be named after the generic is specified.
# destination addresses for random message generation
#"top.ring0.f0.llc_src.low dest address" :
#"top.ring0.f0.llc_src.high dest address" :
```

```

".ring0.*.llc_src.low dest address": 10
".ring0.*.llc_src.high dest address": 19
".ring1.*.llc_src.low dest address": 0
".ring1.*.llc_src.high dest address": 19

# range of priority values that can be assigned to packets; FDDI
# standards allow for 8 priorities of asynchronous traffic. MIL3's
# original model is modified to allow each station to generate multiple
# priorities, within a specified range.
".*.llc_src.high pkt priority" : 7
".*.llc_src.low pkt priority" : 0

# arrival rate(frames/sec), and message size (bits) for random message
# generation at each station on the ring.
".*.*.arrival rate" : 250
".*.*.mean pk length" : 20000
#"ring0.f9.*.arrival rate": 0
#"ring0.f9.*.mean pk length": 0
#"ring1.f9.*.arrival rate": 0
#"ring1.f9.*.mean pk length": 0

# 7APR94 - S.Karayakaylar
# determine which load balancing algorithm is in use in the CPNI
# User should specify the algorithm before simulation.
#
# 0 (zero) ----> circular load balancing algorithm (default)
# 1 (one) ----> empty allocation algorithm
#
"top.ring0.f9.llc_sink.load balancing algorithm": 0

# 15APR94 :determine the station address of the network interfaces in
# both rings.
# CPNI
"top.ring0.f9.llc_sink.station_address": 9
# SPNI
"top.ring1.f9.llc_sink.station_address": 19

# 12DEC93: total offered load is the sum of all stations' loads (Mbps).
# Compute this by hand; this value is useful for generating
# scalar plots where offered load is the abscissa.
total_offered_load_0 : 50
asynch_offered_load_0 : 45
total_offered_load_1 : 50
asynch_offered_load_1 : 45

# set the proportion of asynchronous traffic
# a value of 1.0 indicates all asynchronous traffic
".*.*.asynch_mix" : 0.9

```



```

*** Ring configuration attributes used by "fddi_mac" ***

# allocate percentage of synchronous bandwidth to each station
# this value should not exceed 1 for all stations combined; OPNET does not
# enforce this; 01FEB94: this must be less than 1; see equation below
"*.*.mac.sync bandwidth" : 0.08955675
#"top.ring0.f9.mac.sync bandwidth": 0.0

# Target Token Rotation Time (one half of maximum synchronous response time)
# (This is commented out for compatibility with the fddi_script, which
# sets T_Req on the simulation command line; remove the comment pound-sign
# below to make this environment file self-sufficient.)

#  $SUM(SA_i) + D_{Max} + F_{Max} + Token\_Time \leq TTRT$ 
# Powers gives TTRT = 10 ms as necessary for voice transmission; in "BONeS",
#  $D_{Max} + F_{Max} + Token\_Time = 1.97888$  ms.
"*.*.mac.T_Req" : .004


# Index of the station which initially launches the token
# 17APR94 : -Karayakaylar
# This index should be greater than the maximum station number
# Bridge stations spawns token for interconnected simulation by default.
"spawn station": 20


# Delay incurred by packets as they traverse a station's ring interface
# see Powers, p. 351 for a discussion of this (Powers gives 1usec,
# but 60.0e-08 agrees with Dykeman & Bux)
station_latency: 60.0e-08


# Propagation Delay separating stations on the ring.
# If propagation delay is 5.085 microsec/km, this corresponds to
# to a 50 station ring with a circumference of 50 km.
# (The value given for propagation delay corresponds to Powers, and to
# Dykeman & Bux)
prop_delay: 5.085e-06


# CDL link related attributes -Karayakaylar 7APR94
# The attributes below are specified with respect to the jammer type
# There are two types of jamming models which the CDL is exposed to.
#
# (1) Pulsed jammer (jammer_type = 0)
# (2) Channel-swept jammer (jammer_type = 1)
#
# NOTE: For pulsed jammer init_jam_offset may be zero, whereas a proper
# offset should be specified for channel-swept jammer.

```

```

# jam_length, jam_ber, interval_bet_jam_len, ber_bet_jam_len are maximum
# values in the case of pulsed jammer since they are randomized in the
# error allocation pipeline stage.
# For channel-swept jammer only jam_ber and ber_bet_jam_len attributes are
# maximum values to be randomized.
# return link ls_0 to ls_3
# command link ls_4
*.ls_0.jam_length: 0.05
*.ls_1.jam_length: 0.02
*.ls_2.jam_length: 0.01
*.ls_3.jam_length: 0.09
*.ls_4.jam_length: 0.06
*.ls_0.jam_ber: 2e-3
*.ls_1.jam_ber: 2e-3
*.ls_2.jam_ber: 2e-3
*.ls_3.jam_ber: 2e-3
*.ls_4.jam_ber: 0.0
*.ls_0.interval_bet_jam_len: 0.03
*.ls_1.interval_bet_jam_len: 0.03
*.ls_2.interval_bet_jam_len: 0.03
*.ls_3.interval_bet_jam_len: 0.03
*.ls_4.interval_bet_jam_len: 0.03
*.ls_0.ber_bet_jam_len: 2e-6
*.ls_1.ber_bet_jam_len: 2e-6
*.ls_2.ber_bet_jam_len: 2e-6
*.ls_3.ber_bet_jam_len: 2e-6
*.ls_4.ber_bet_jam_len: 0.0
*.ls_0.init_jam_offset: 0.0
*.ls_1.init_jam_offset: 0.0
*.ls_2.init_jam_offset: 0.0
*.ls_3.init_jam_offset: 0.0
*.ls_4.init_jam_offset: 0.0
*.ls_0.jammer_type: 0
*.ls_1.jammer_type: 0
*.ls_2.jammer_type: 0
*.ls_3.jammer_type: 0
*.ls_4.jammer_type: 0

# Return and command link propagation delays are specified as 60 msec.
*.ls_0.delay: 0.06
*.ls_1.delay: 0.06
*.ls_2.delay: 0.06
*.ls_3.delay: 0.06
*.ls_4.delay: 0.06

*** Simulation related attributes

# Token Acceleration Mechanism enabling flag.
# It is recommended that this mechanism be enabled for most situations
# 16APR94 : for bridged fddi cdl_interconnection network this flag
# must be zero. Otherwise, program fault occurs. -Karayakaylar

```

accelerate\_token: 0

# Run control attributes

seed:

10

duration: 1

verbose\_sim: TRUE

upd\_int: .1

os\_file: cd14\_lb0jam0

# (This is commented out for compatibility with the fddi\_script, which  
# sets the output vector file on the simulation command line; remove the  
# comment pound-sign below to make this environment file self-sufficient.)

ov\_file: cd14\_lb0jam0

# Opnet Debugger (odb) enabling attribute

# debug: TRUE



# APPENDIX I

## SAMPLE ENVIRONMENT FILE EXCERPT FOR CHANNEL-SWEPT JAMMER

```
# cdl4_lb1jam1.ef
# sample simulation configuration file for
# two interconnected 10 station network in the
# existence of channel-swept jammer interference (137.088 Mbps channel hierarchy )
# with empty selection load balancing algorithm
. . .
. . .
. . .
. . .
. . .
# CDL link related attributes -Karayakaylar 7APR94
# The attributes below are specified with respect to the jammer type
# There are two types of jamming models which the CDL is exposed to.
#
# (1) Pulsed jammer          (jammer_type = 0)
# (2) Channel-swept jammer   (jammer_type = 1)
#
# NOTE:For pulsed jammer init_jam_offset may be zero, whereas a proper
# offset should be specified for channel-swept jammer.
# jam_length, jam_ber, interval_bet_jam_len, ber_bet_jam_len are maximum
# values in the case of pulsed jammer since they are randomized in the
# error allocation pipeline stage.
# For channel-swept jammer only jam_ber and ber_bet_jam_len attributes are
# maximum values to be randomized.
# return link ls_0 to ls_3
# command link ls_4
*.ls_0.jam_length:                0.02
*.ls_1.jam_length:                0.02
*.ls_2.jam_length:                0.02
*.ls_3.jam_length:                0.02
*.ls_4.jam_length:                0.02
*.ls_0.jam_ber:                   2e-3
*.ls_1.jam_ber:                   2e-3
*.ls_2.jam_ber:                   2e-3
*.ls_3.jam_ber:                   2e-3
*.ls_4.jam_ber:                   0.0
*.ls_0.interval_bet_jam_len:      0.06
*.ls_1.interval_bet_jam_len:      0.06
*.ls_2.interval_bet_jam_len:      0.06
*.ls_3.interval_bet_jam_len:      0.06
*.ls_4.interval_bet_jam_len:      0.06
*.ls_0.ber_bet_jam_len:           2e-6
*.ls_1.ber_bet_jam_len:           2e-6
```

*.ls_2.ber_bet_jam_len:		2e-6
*.ls_3.ber_bet_jam_len:		2e-6
*.ls_4.ber_bet_jam_len:		0.0
*.ls_0.init_jam_offset:		0.0
*.ls_1.init_jam_offset:		0.02
*.ls_2.init_jam_offset:		0.04
*.ls_3.init_jam_offset:		0.06
*.ls_4.init_jam_offset:		0.0
*.ls_0.jammer_type:	1	
*.ls_1.jammer_type:	1	
*.ls_2.jammer_type:	1	
*.ls_3.jammer_type:	1	
*.ls_4.jammer_type:	1	
. . .		
. . .		
. . .		
. . .		
. . .		
. . .		

## LIST OF REFERENCES

1. Nix, E. E., "Modeling and Simulation of a Fiber Distributed Data Interface Local Area Network (FDDI LAN) Using OPNET for Interfacing Through the Common Data Link (CDL)," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1994.
2. ISO/IEC 10038:ANSI/IEEE, Std 802.1D, International Organization for Standardization, July 1993.
3. Seifert, W. M., "Bridges and Routers," *IEEE Network*, vol. 2, no. 1, pp. 57-64, Jan. 1988.
4. Perlman, R., *Interconnections*, Reading, MA: Addison-Wesley, 1992.
5. "Standards Project: Remote MAC Bridging," Institute of Electrical and Electronics Engineers, Draft-P802.1G/D8, Mar. 1993.
6. Simpson, W., "The Point-to-Point Protocol (PPP)," Request for Comments (RFC) 1548, Dec. 1993.
7. Takeshi, A., "Distributed Multilink System for Very-High-Speed Data Link Control," *IEEE J. on Selected Areas in Commun.*, vol. 11, no. 4, pp. 540-549, May 1993.
8. Marsden, P., "Internetworking IEEE 802/FDDI LAN's via the ISDN Frame Relay Bearer Service," *Proc. of the IEEE*, vol. 79, no. 2, Feb. 1991.
9. Mongiovi, L., "A Proposal for Interconnecting FDDI Networks Through B-ISDN," *Proc. of INFOCOM*, pp. 1160-1167, June 1991.





## INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library Code 052 Naval Postgraduate School Monterey, CA 93943-5000	2
3. Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4. Professor Shirdhar B. Shukla, Code EC/Sh Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	2
5. Professor Gilbert Lundy, Code CS/Ln Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5118	2
6. Professor Paul Moose, Code EC/Me Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93940-5121	1
7. Mark Russon, UNISYS Mail Station F2-G14 640 North 2000 West Salt Lake City, UT 84116-2988	1
8. CDL Program Manager Defense Support Project Office Washington DC 20330-1000	1

		No. Copies
9.	Deniz Kuvvetleri Komutanligi Personel Daire Baskanligi Bakanliklar, Ankara, Turkey	2
10.	Golcuk Tersanesi Komutanligi Golcuk, Kocaeli, Turkey	1
11.	Deniz Harp Okulu Komutanligi 81704 Tuzla, Istanbul, Turkey	1
12.	Taskizak Tersanesi Komutanligi Kasimpasa, Istanbul, Turkey	1
13.	Selcuk Karayakaylar Kardesler sk. Huzur Apt. A-Blok 11/10 80700 Dikilitas, Istanbul, Turkey Department of Electrical and Computer Engineering	1